Linter Standard
Linter Bastion
Linter RealTime
Linter Multiversion

Процедурный язык

НАУЧНО-ПРОИЗВОДСТВЕННОЕ ПРЕДПРИЯТИЕ

Товарные знаки

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

Интеллектуальная собственность

Правообладателем продуктов ЛИНТЕР®, НЕВОД®, LAB™, ЛАКУНА является компания РЕЛЭКС (1990–2011). Все права защищены. Данный документ является собственностью компании РЕЛЭКС. Ни одна часть данного документа не может быть воспроизведена, передана, преобразована, сохранена в системе поиска информации, переведена на другой язык или компьютерный язык в какой-либо форме, какими-либо средствами, электронными, механическими, магнитными, оптическими, химическими, ручными или иными, без предварительного разрешения компании РЕЛЭКС.

О документе

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

Адрес

394006, г. Воронеж, ул. 20-летия Октября, 119.
Тел./факс: (473) 2-711-711, 2-778-333.
e-mail: market@relex.ru.

Адрес для корреспонденции

394000, г. Воронеж, а/я 137.

Техническая поддержка

Отдел поддержки и сопровождения программных продуктов:

телефон: (473) 2-711-711 с 9:00 до 18:00 мск.
e-mail: support@relex.ru, market@relex.ru.

С целью повышения качества разрабатываемых программных средств и предоставляемых услуг в компании РЕЛЭКС действует автоматизированная система учёта и обработки рекламаций. Обо всех обнаруженных недостатках и ошибках в программном продукте и/или документации на него просим сообщать нам на Internet–странице рекламация.

Оглавление

Предисловие ... 5

Назначение документа ... 5

Для кого предназначен документ ... 5

Принятые обозначения и соглашения ... 5

Дополнительные документы ... 6

Общие сведения ... 7

Элементы языка ... 8

Имена (идентификаторы) ... 8

Комментарии ... 8

Типы данных ... 8

Совместимость типов данных ... 9

Константы ... 10

Числовые константы ... 10

Символьные константы ... 11

UNICODE-константы ... 11

Константы типа DATE ... 11

Логические константы ... 12

Курсорные константы ... 12

NULL-значения ... 12

Определение переменных ... 12

Предопределенные переменные триггера ... 12

Старое и новое значение поля ... 12

Триггерные предикаты ... 13

Количество обработанных строк ... 14

Идентификатор канала ... 14

Общий вид хранимой процедуры ... 15

Формат заголовка ... 15

Формат блока описаний ... 17

Формат блока кода ... 18

Формат блока обработки исключений ... 18

Операторы ... 21

Оператор – выражение ... 21

Условный оператор ... 22

Оператор выбора ... 22

Оператор цикла ... 23

Безусловный переход ... 24

Вызов хранимой процедуры ... 24

Возврат из процедуры ... 25

Вызов исключения ... 25

Передача исключения ... 25

Открытие курсора ... 26

Выборка данных из открытого курсора ... 27

Закрытие курсора ... 27

Выполнение некурсорного запроса ... 28

Завершение транзакции ... 29

Запросы ...29
Претранслируемые запросы ... 29
Динамические запросы ... 30
Выражения ...30
Операнды ... 31
Операции... 32
Операции в числовых выражениях ...32
Операции в символьных выражениях...32
Операции в логических выражениях...32
Операции в выражениях типа «дата» ...33
Присвоение значений ...33
Условные выражения ...34
Пакетное добавление ...34
Работа с типом данных BLOB ...38
Добавление значения в конец BLOB ...39
Чтение значения из BLOB на общем уровне ...40
Чтение int-значения из BLOB ...40
Чтение smallint-значения из BLOB...41
Чтение bigint-значения из BLOB ...41
Чтение real-значения из BLOB ...41
Чтение numeric-значения из BLOB ...42
Чтение double-значения из BLOB ...42
Чтение char-значения из BLOB ...43
Чтение nchar-значения из BLOB ...43
Чтение date-значения из BLOB ...43
Чтение bool-значения из BLOB ...44
Установка текущей позиции для чтения BLOB ...44
Получение размера BLOB-данных...44
Установка текущего BLOB-столбца...45
Типизация BLOB-столбца...45
Поддержка кодовых страниц...46
Стандартные функции ...47
Символьные функции ... 47
Дополнение строки слева...47
Дополнение строки справа...48
Выделение конечной подстроки ...49
Дублирование строки...50
Поиск подстроки...50
Определение длины символьной строки ...51
Определение длины строки в байтах ...52
Удаление крайних пробелов из символьной строки ...52
Удаление символов из строки слева ...53
Удаление символов из строки справа...53
Выделение символьной подстроки ...54
Определение позиции подстроки ...55
Замена всех подстрок ...56
Замена символов строки ...57
Преобразование строки...58
Удвоение символа в строке...58
Преобразование символов строки в коды...59
Преобразование из символьного вида в шестнадцатеричный ...59

Преобразование строки к верхнему регистру 59
Преобразование строки к нижнему регистру 60
Перевод начальной буквы слова в заглавную 60
Корректировка подстроки 60
Преобразование числового выражения в символьный вид 61
Фонетический код строки 63
Определение близости фонетического звучания строк 63
Функции для работы с UNICODE-значениями 64
Определение длины UNICODE-строки 64
Удаление пробелов UNICODE-строки 64
Выделение UNICODE–подстроки 65
Дополнение UNICODE–строки слева 65
Дополнение UNICODE–строки справа 65
Выделение последних символов UNICODE–строки 66
Дублирование UNICODE–строки 66
Выделение позиции подстроки 66
Преобразование строки к верхнему регистру 66
Преобразование строки к нижнему регистру 67
Преобразование в UNICODE-строку 67
Удаление символов из UNICODE–строки слева 67
Удаление символов из UNICODE–строки справа 68
Перевод начальной буквы UNICODE–слова в заглавную 68
Корректировка UNICODE–подстроки 68
Замена всех UNICODE–подстрок 68
Замена символов UNICODE–строки 69
Математические функции 69
Вычисление абсолютного значения 69
Округление до целого с избытком 69
Округление до целого с недостатком 70
Округление значения типа «дата-время» 70
Усечение представления значения типа «дата-время» 71
Тригонометрические функции 71
Обратные тригонометрические функции 72
Гиперболические функции 72
Экспоненциальная функция 73
Логарифмические функции 73
Округление с заданной точностью 73
Усечение числа с заданной точностью 74
Определение знака числа 74
Вычисление квадратного корня числа 75
Работы с датами 75
Выделение дня из даты 75
Выделение месяца из даты 75
Выделение года из даты 75
Выделение часа из даты 76
Выделение минут из даты 76
Выделение секунд из даты 76
Выделение тиков из даты 77
Формирование даты 77
Получение текущей даты 77
Последний день месяца 77
Дата очередного дня недели 78
Помесячное изменение даты 79
Выделение заданных элементов даты 79
Изменение даты на заданный интервал времени 80

Вычисление интервала между двумя датами ... 81
Функции преобразования типов ... 82
Представление числа в символьном виде ... 82
Представление числа в символьном виде с учетом знака ... 82
Представление числа в символьном виде с учетом знака и заданной точности ... 83
Представление даты в символьном виде ... 84
Преобразование байтового значения в строку ... 85
Универсальное преобразование в строку ... 85
Преобразование строки в дату ... 85
Преобразование в тип smallint ... 86
Перобразование в тип int ... 87
Преобразование в тип bigint ... 88
Преобразование в тип real ... 88
Преобразование в тип numeric ... 88
Преобразование символьной строки в строку байт ... 88
Преобразование значения в шестнадцатеричное представление ... 89
Функции для работы с курсорами ... 89
Проверка выхода курсора за пределы выборки ... 89
Количество ответов в курсорной выборке, сделанной по курсору ... 90
Определение кода завершения для курсора ... 90
Определение номера текущей строки курсора ... 91
Транзакции в процедурах ... 91
Общие положения ... 91
Вложенные транзакции ... 91
Инициирование транзакции ... 92
Подтверждение транзакции ... 93
Откат транзакции ... 93
Логические функции ... 93
Логическое “И” ... 93
Логическое “ИЛИ” ... 94
Прочие функции ... 94
Нахождение максимального числа ... 94
Вычисление остатка от деления ... 94
Генерация псевдослучайного числа ... 95
Инициализация датчика случайных чисел ... 95
Определение текущего пользователя ... 95
Получение имени базы данных ... 96
Преобразование строки по алгоритму md5 ... 96
Генерация пользовательского кода завершения ... 97
Момент срабатывания триггера ... 97
Приостанов выполнения процедуры ... 99

Предметный указатель ... 101

Указатель операторов ... 102

Указатель функций ... 103

Предисловие

Назначение документа

Документ содержит описание процедурного языка СУБД ЛИНТЕР, предназначенного для программирования хранимых процедур и триггеров базы данных. Язык разработан в соответствии с рекомендациями стандарта SQL-92/PSM (Persistent Stored Modules).

Документ может использоваться для работы с любой версией СУБД ЛИНТЕР. Особенности конкретных версий оговариваются по тексту.

Для кого предназначен документ

Документ предназначен для программистов, разрабатывающих приложения для работы с СУБД ЛИНТЕР.

Принятые обозначения и соглашения

Обозначение	Пример	Значение
Курсив	*Растровым* называется изображение…	Новый термин в тексте
Полужирный шрифт	В этом случае необходимо переносить **все** физические файлы.	Выделение в тексте
Подчеркнутый шрифт	Подробную информацию о работе программы можно получить на сайте www.dmk.ru.	Адреса страниц Internet
Текст, разделенный знаком ⇒	Выполните команду **View ⇒ Properties** (Вид ⇒ Свойства).	Последовательность выполнения команд
Текст, заключенный в < >, со знаком + между ними	<Ctrl>+<C>	В < > заключаются клавиши клавиатуры, знак + означает сочетание клавиш
Крупный моноширинный текст	`SQL> _q`	Текст командной строки

Мелкий моноширинный текст	`Page`
	`Time`
	`Count`
		Текст программы

Заглавные буквы	BROWSE	Названия команд, слова, зарезервированные в SQL, ключевые слова
Курсив в < >	<return statement>	Определяемый элемент синтаксической конструкции
Символ ::=		Равенство по определению. Слева от знака стоит определяемое понятие, справа – собственно определение понятия
Квадратные скобки [ ]	`DBSTORE [-d -n -o -p -r`	Необязательные элементы

Обозначение	Пример	Значение
	`-t -u]`	конструкции. В данном примере ключи не являются обязательными элементами команды
Вертикальная черта |	<return value> ::= `<value expression>|NULL`	Указывает на то, что все предшествующие ей элементы списка являются необязательными и могут быть заменены любым другим элементом списка после этой черты
Фигурные скобки { }	`CODEPAGE {866 | 1251 | KOI8}`	Указывают на то, что все, находящееся внутри них, является единым целым
Многоточие «...»	`Характеристики столбца MAKE CHAR(20) MODEL CHAR(20) ... SQL>`	Означает, что предшествующая часть может быть повторена любое количество раз
Многоточие, внутри которого находится запятая «.,..»		Указывает на то, что предшествующая часть оператора, состоящая из нескольких элементов, разделенных запятыми, может иметь произвольное число повторений
Текст со знаком 🗎 на сером фоне	🗎 Если конфигурация страницы-шаблона не учитывала свойств, команда будет выполнена некорректно.	Примечание

Дополнительные документы

- СУБД ЛИНТЕР. Справочник по SQL.
- СУБД ЛИНТЕР. Встроенный SQL.
- СУБД ЛИНТЕР. Системные таблицы.

Общие сведения

Процедурный язык предназначен для написания текстов триггеров и хранимых процедур.

Все лексические элементы (лексемы) в тексте на процедурном языке могут разделяться любым количеством пробелов, табуляций, символов новой строки и комментариев, которые игнорируются.

Все ключевые слова и идентификаторы в тексте на процедурном языке не чувствительны к регистру, то есть, например, любое из слов INTEGER, integer или Integer воспринимается однозначно как одно ключевое слово. Исключение составляют имена других процедур, вызываемых из процедуры. Для них действует общее правило для имен в СУБД ЛИНТЕР: если имя написано без двойных кавычек, регистр не имеет значения, если же имя взято в кавычки, подразумевается в точности то, что написано. В данном документе для выделения ключевых слов везде используется верхний регистр.

Элементы языка

Имена (идентификаторы)

Имена (идентификаторы) используются для именования передаваемых параметров, локальных переменных, исключений, меток исполняемых операторов. Имя может состоять из последовательности любых алфавитно-цифровых и графических символов, за исключением нижеследующих:

+ - * / ^ ( ) \ = # < > . , ; [ ] { } \ " $:

Имя не должно начинаться с цифры и может содержать до 30 символов.

Комментарии

Комментарием является весь остаток строки после символов «//», если они встречаются в строке.

Типы данных

В процедурном языке СУБД ЛИНТЕР доступны следующие типы данных:

1) BIGINT – используется для представления целых знаковых чисел в диапазоне от –9 223 372 036 854 775 808 до +9 223 372 036 854 775 807.
2) INTEGER (INT) – используется для представления целых знаковых чисел в диапазоне от –2 147 483 648 до +2 147 483 647.
3) SMALLINT – используется для представления целых знаковых чисел в диапазоне от –32 768 до +32 767.
4) REAL – используется для представления чисел с плавающей точкой одинарной точности (значения в диапазоне от -1.0E+38 до +1.0E+38, точность – 6 значащих цифр).
5) DOUBLE – используется для представления чисел с плавающей точкой двойной точности (значения от -1.0E+38 до +1.0E+38, точность – 15 значащих цифр).
6) NUMERIC – используется для представления чисел с фиксированной точкой.
7) CHAR (<размер>) – используется для представления алфавитно-цифровых символов, графических знаков и управляющих кодов в виде текстовой строки фиксированной длины (до 3919 знаков).

Максимальная длина строковых типов данных процедурного языка отличается от максимальной длины строковых типов данных языка SQL СУБД ЛИНТЕР.

8) VARCHAR (<размер>) – используется для представления алфавитно-цифровых символов, графических знаков и управляющих кодов в виде текстовой строки переменной длины (до 3919 знаков).
9) BYTE (<размер>) – используется для представления алфавитно-цифровых символов, графических знаков и управляющих кодов в виде шестнадцатеричной строки фиксированной длины (до 3919 знаков).
10) VARBYTE (<размер>) – используется для представления алфавитно-цифровых символов, графических знаков и управляющих кодов в виде шестнадцатеричной строки переменной длины (до 3919 знаков).

11) NCHAR (<размер>) – используется для представления UNICODE–строки фиксированной длины (до 1959 знаков) (только для версии СУБД ЛИНТЕР 6.0 и выше).
12) NVARCHAR (<размер>) – используется для представления UNICODE–строки переменной длины (до 1959 знаков) (только для версии СУБД ЛИНТЕР 6.0 и выше).
13) DATE – используется для представления информации о дате и времени. Информация задается в формате:
    DD.MM.YYYY:[HH:[MI:[SS:[TT]]],
    где:

DD	день месяца	(от 1 до 31)
MM	месяц года	(от 1 до 12)
YYYY	год	(от 1 до 9999)
HH	часы	(от 0 до 23)
MI	минуты	(от 0 до 59)
SS	секунды	(от 0 до 59)
TT	тики	(от 0 до 99)

Допустима нулевая дата 00.00.0000:00:00:00:00

14) BOOL – используется для представления логической информации. Допустимыми значениями являются TRUE и FALSE.
15) CURSOR (<имя> <скалярный тип> [,<имя> <скалярный тип>]...[,<имя> <скалярный тип>]) – используется для задания структуры данных, соответствующей полям таблицы или представления базы данных.
16) BLOB – поддержка BLOB–данных выполняется с помощью встроенных функций.
17) для указания типа может использоваться конструкция TYPEOF(<имя объекта>), где <имя объекта> - имя любой переменной, столбца, таблицы, синонима представления или ключевое слово RESULT. TYPEOF(RESULT) означает тип результата процедуры.

Пример

```
typeof(SYSTEM.AUTO.NAME);
typeof(”Склад”.”Код_товара”);
```

```
create or replace procedure prc_test01(in n1 char(3)) result char(15) for debug
declare
   var i, j, k, l typeof(aaa.i); //
   var c cursor(i typeof(aaa.i), j typeof(aaa.i), k typeof(aaa.i)); //
code
    open c for direct "select 'bbb', 'ccc', i from aaa;"; //
    l := c.k; //
    fetch c into i, j, k; //
    return n1 + i + j + l + k; //
end;
```

Совместимость типов данных

Для символьных выражений, дат и логических типов данных совместимы только сами эти типы. Для числовых выражений совместимы любые числовые типы.

Константы

Числовые константы

Константы числовых типов записываются общепринятым способом. Для целых чисел – это последовательность цифр, перед которой может стоять знак '+' или '–'. Для чисел с плавающей точкой – аналогично, но между цифрами может также находиться точка (символ «.»), отделяющая целую часть от дробной. Точка может находиться в конце, указывая на число с нулевой дробной частью, но не может стоять в начале последовательности (ноль для целой части пишется всегда явно).

Константы целого типа по умолчанию приводятся к следующим типам:

Значение константы	Тип по умолчанию
от –32 768 до +32 767	SMALLINT
от –2 147 483 648 до –32 768	INTEGER
от +32 767 до +2 147 483 647	
от –9 223 372 036 854 775 808 до –2 147 483 648	BIGINT
от +2 147 483 647 до +9 223 372 036 854 775 807	

Константы с десятичной точкой по умолчанию приводятся к типу DECIMAL.

Для указания конкретного типа, к которому должна быть приведена константа, применяется суффикс, который добавляется в конец константы:

Суффикс	Приводимый тип
I	INTEGER
B	BIGINT
N	DECIMAL
R	REAL
D	DOUBLE

Примеры числовых констант

```
0

+123I

-456

3.1415

-0.33R

777B
```

Символьные константы

Константы символьного типа записываются в двойных кавычках полностью, аналогично языку программирования C/C++. Они могут содержать печатные символы и специальные слеш-последовательности:

- \n – символ новой строки;
- \r – символ возврата каретки;
- \t – символ табуляции;
- \x<код> – символ с указанным (шестнадцатеричным) кодом;
- \<символ> – указанный символ. Может использоваться для экранирования кавычек и самого символа «\».

Длинные константы можно разбивать на части, размещаемые на отдельных строках текста, т.е. конструкция типа:

```
"aaa"
"bbb"
"ccc"
```

распознается как одна символьная константа "aaabbbccc", которую разбили на несколько строк для удобочитаемости (так же, как это делают компиляторы C/C++).

Примеры символьных констант

```
"abcd"

"New Line\n"

"\x7 the Bell"

"\"c:\\linter\""
```

UNICODE-константы

Чтобы задать константу, представленную как UNICODE-значение, необходимо использовать обычную символьную константу, которая преобразуется к типу NCHAR при помощи функции tonchar. При этом кодировка символьной константы соответствует кодировке, в которой работало клиентское приложение на момент создания процедуры (триггера), и преобразование в UNICODE будет выполняться именно с учетом этой кодировки.

Константы типа DATE

Константы типа DATEимеют формат DD.MM.YYYY[:HH[:MI[:SS[.TT]]]]. Это означает, что любая конечная последовательность, т.е. :HH:MM:SS.TT, :MM:SS.TT, :SS.TT или .TT, может отсутствовать. Каждая последовательность символов DD, MM и так далее означает от одной до двух (четырех для YYYY) цифр:

- DD – номер дня;
- MM – номер месяца;
- YYYY – номер года;
- HH – часы;
- MI – минуты;
- SS – секунды;
- TT – тики (сотые доли секунд).

Примеры констант типа DATE

```
21.08.1997

21.8.1997:20:53

21.08.1997:9:10.55
```

Логические константы

Константы типа BOOL– это ключевые слова TRUE и FALSE.

Курсорные константы

Констант типа CURSOR нет.

NULL-значения

В хранимых процедурах объекты скалярного типа могут иметь значение NULL, интерпретируемое как отсутствие данных. NULL-значение можно присваивать переменной любого типа, а так же сравнивать текущие значения переменных с NULL-значением. Переменные могут получить NULL-значение в результате явного присвоения или как значение по умолчанию (см. ниже).

При попытке выполнения любых действий (арифметических, логических и т.д.) с NULL-значениями, кроме операций присвоения и сравнения, происходит исключение NULLDATA (см. ниже).

Для указания NULL-значения используется константа NULL.

Определение переменных

Описание переменных аналогично описанию группы параметров. Формат определения см. на стр. 17.

Предопределенные переменные триггера

Старое и новое значение поля

Внутри тела триггера можно использовать специальные предопределенные переменные. Эти переменные имеют значение обрабатываемой записи до и после выполнения триггерной операции. С точки зрения кода триггера эти значения имеют тип CURSOR, то есть для обращения к полю обрабатываемой записи используется конструкция <имя значения>.<имя поля>. <Имя поля> соответствует имени поля таблицы. <Имя значения> – это то, что указано во фразе OLD [AS] и NEW [AS] предложения CRETAE TRIGGER, или OLD и NEW для старого и нового значения записи соответственно, если OLD [AS] и NEW [AS] не заданы.

Внутри кода триггера, вызываемого перед модификацией строки таблицы, можно присваивать новые значения полям переменной NEW (или той, что указана в NEW AS). В этом случае они будут использоваться для формирования значения обрабатываемой записи как результата выполнения операции, с которой связан триггер.

Значения OLD и NEW определены не для всех триггеров. Так, в триггере на INSERT определено только значение NEW, а в триггере на DELETE – только OLD. В триггерах же на весь STATEMENT эти значения вообще не определены.

Пример

```
if old.personid <> new.personid then
  execute direct "select personid from person where"
  "personid="+ itoa(new.personid) + ";"; //
  if errcode() = 2 then
    execute direct "insert into journal values ('AUTO',
    'UPDATE',"+itoa(old.personid) + ",sysdate,'update"
    "to bad personid - IGNORED');"; //
    return false; //
  endif
endif
```

Триггерные предикаты

Для триггера могут быть назначены несколько условий активизации (через логическую операцию ИЛИ (OR)). В этом случае узнать, какое конкретно событие активизировало триггер, можно с помощью триггерных предикатов INSERTING, DELETEING, UPDATING. Триггерные предикаты – это предопределенные переменные, которые, в зависимости от события, принимают одно из значений TRUE или FALSE. С их помощью можно значительно сэкономить на кодах триггера. Например, вместо трех индивидуальных триггеров на разные события можно написать один триггер, который, в зависимости от события, будет выполнять ту или иную операцию.

Триггерный предикат	Принимаемое значение
INSERTING	TRUE, если триггер активизирован оператором INSERT, иначе – FALSE
UPDATING	TRUE, если триггер активизирован оператором UPDATE, иначе – FALSE
DELETING	TRUE, если триггер активизирован оператором DELETE, иначе – FALSE

Триггерные предикаты позволяют определить тип события, на которое сработал триггер, но не момент срабатывания триггера: до (BEFORE), после (AFTER) или вместо (INSTEAD OF) произошедшего события. Для уточнения момента срабатывания триггера используется функция sysevent() процедурного языка.

Пример

```
create or replace table test(ch char(10));
create or replace table result(ch char(10));

create or replace trigger t1_aaa before insert or update or delete on test
for each row execute
declare
code
   if ( inserting ) then
     execute direct "insert into result values ('inserting');"; //
   endif; //
   if ( updating ) then
     execute direct "insert into result values ('updating');"; //
   endif; //
   if ( deleting ) then
     execute direct "insert into result values ('deleting');"; //
```

```
      endif; //
      return true; //
   end;

   insert into test values('value 1');
   update test set ch = 'value 2' where ch = 'value 1';
   delete from test where ch = 'value 2';
   select * from result;
   CH
    --
   |inserting |
   |updating  |
   |deleting  |
```

Количество обработанных строк

Предопределенная переменная ROWCOUNT содержит количество строк, реально обработанное последней командой. Триггер запускается не при попытке изменить конкретную строку, а в момент выполнения команды изменения.

Для операций DELETE и UPDATE (в том числе для каскадных операций), а также для операции INSERT FROM SELECT переменная ROWCOUNT содержит общее количество удалённых или модифицированных записей (для каскадных операций – количество удалённых/модифицированных записей на текущем уровне), для операций INSERT, DELETE CURRENT, UPDATE CURRENT переменная содержит значение 1.

Пример

Следует различать предопределенную переменную ROWCOUNT и функцию rowcount() (которая возвращает количество ответов в курсоре).

```
create or replace table test(i int);
insert into test values(1);
insert into test values(2);
create or replace table test_result(ch char(20));
create or replace trigger test_tr before update on test for each statement old as
"OLD" new as "NEW" execute FOR DEBUG
code
  execute direct "insert into test_result(ch) values('" + itoa(rowcount) + "');";
//
  execute direct "select make from auto;"; //
  execute direct "insert into test_result(ch) values('" + itoa(rowcount()) +
"');"; //
end;
update test set i = 100;
select * from test_result;

|2            |
|1000         |
```

Идентификатор канала

Псевдопеременная SESSIONID содержит идентификатор канала, в котором был активизирован триггер или запущена на выполнение хранимая процедура. Для вложенных вызовов процедур и триггеров в качестве SESSIONID используется SESSIONID верхнего канала.

Идентификатор канала создается ядром СУБД при выполнении пользователем соединения с БД (открытия канала) и остается неизменным для всех подканалов и курсоров, порождаемых при выполнении пользовательских SQL–запросов, триггеров и хранимых процедур по этому соединению.

Тип значения псевдопеременной SESSIONID – BIGINT.

Значение идентификатора соответствует полю SESSIONID из системной таблицы $$$CHAN и псевдозначению SESSIONID в SQL-запросах.

Пример

Пример использования псевдопеременной SESSIONID см. в описании функции sysevent().

Общий вид хранимой процедуры

Хранимая процедура записывается в виде процедурного блока, который имеет следующий формат:

```
<заголовок><тело процедуры>
<тело процедуры>::=
[<блок описаний>]
<блок кода>
[<блок обработки исключений>]
END
```

Формат заголовка

Заголовок хранимой процедуры имеет следующий формат:

```
PROCEDURE <имя>(<список параметров>)
[AUTHID { CURRENT_USER |  DEFINER}]
[RESULT <тип>] [FOR DEBUG]
```

Список параметров может быть пустым или содержать одну либо более группу параметров. Группы параметров разделяются точкой с запятой. Группа параметров имеет формат:

```
<модификатор> <список имен><тип> [DEFAULT <список инициализаторов>]
```

Модификатор – это одно из слов IN, OUT или INOUT, которые указывают, соответственно, на то, что значение параметра передается в процедуру (используется только как входной параметр), возвращается из процедуры (используется только как выходной параметр) или передается как туда, так и обратно (используется как входной/выходной параметр).

Список имен содержит имена параметров, разделенные запятой (или одно имя).

Список инициализаторов представляет собой список выражений, разделенных запятыми. Выражения для начальных значений должны быть такими, чтобы их можно было вычислить на этапе трансляции. Следовательно, для написания этих выражений можно использовать константы и ранее определенные в процедуре параметры и/или переменные

(в последнем случае в качестве значений переменных берутся их значения по умолчанию).

Количество выражений в списке может быть меньше количества параметров в группе. Оставшиеся параметры инициализируются так же, как при отсутствии фразы DEFAULT, т.е. NULL-значениями (отсутствие DEFAULT равносильно явной записи DEFAULT NULL).

Опция AUTHID CURRENT_USER заставляет выполнять процедуру в контексте текущего пользователя, а опция AUTHID DEFINER –в контексте ее владельца.

Пример

```
  drop user USER1 cascade;
  create user USER1 identified by '';
  grant dba to USER1;
  create or replace user USER2 identified by '';
  grant dba to USER2;

  username USER1/
  create or replace table "TEST" ("ID" int, "TEXT" char(10));
  insert into TEST values( 1, 'aaa');
  create or replace procedure TEST_PROC(IN id INTEGER; IN text CHAR(10))
AUTHID DEFINER result integer
  code
  EXECUTE DIRECT "INSERT INTO USER1.TEST values("+ITOA(id)+",'"+text+"')";//
  return errcode(); //
  end;

  create or replace procedure TEST_UFN(in i int ) AUTHID DEFINER result
cursor( i int )
  declare
  var b typeof(result); //
  code
  open b for direct "select ID from USER1.TEST where ID=" + itoa(i) + ";";
//
  return b; //
  end;

  grant execute as owner on USER1.TEST_PROC to USER2;
  grant execute as owner on USER1.TEST_UFN to USER2;

  username USER2/
  select i from USER1.TEST_UFN(1);
  execute USER1.TEST_PROC(2,'bbb');
```

Предпоследний запрос возвращает одну запись, последний – успешно выполняется. Если же убрать AUTHID DEFINER из текста процедур, то предпоследний запрос вернет 0 записей, а последний – код завершения 1022 (нарушение привилегий).

Все процедуры возвращают некоторое значение (код завершения или возвращаемое значение). Тип этого значения определяется во фразе RESULT. Если она не задана, по умолчанию предполагается NULL.

Если указана фраза FOR DEBUG, процедура транслируется с отладочной информацией, иначе отладочная информация не включается в оттранслированный код процедуры, и процедуру нельзя будет отлаживать отладчиком хранимых процедур.

Фраза DEFAULT используется для пропущенных параметров в конце списка (если при вызове список параметров короче, чем в объявлении процедуры) или в любом месте списка (в этом случае пропущенные параметры заменяются запятыми).

Пример заголовка хранимой процедуры

```
procedure retcur(in name char(20) default "AUTO";
out success bool)
result cursor(i int,
              a char(20),
              s smallint,
              d date,
              n numeric,
              r real
              ) for debug
```

Формат блока описаний

Блок описаний хранимой процедуры имеет следующий формат:

DECLARE
<описание 1>
. . . .
<описание n>

Здесь <описание …> – это описание локальных переменных или описание исключения.

Описание переменных

Описание переменных аналогично описанию группы параметров:

VAR <список имен> <тип> [**DEFAULT** <список инициализаторов>];

Описание исключения

Описание исключения имеет вид:

EXCEPTION <имя> **FOR** <вид исключения>;

Виды исключений следующие:

- DIVZERO – деление на ноль;
- UNDEFPROC – попытка вызова неопределенной процедуры;
- BADPARAM – неверный параметр для процедуры или стандартной функции;
- BADRETVAL – недопустимый тип возвращаемого значения;
- NULLDATA – попытка выполнить недопустимую операцию с NULL-значением (вычисление выражения при отсутствии данных);
- BADCURSOR – несоответствие структуры курсора количеству и/или типам ответов на запрос (при открытии или возврате курсора);
- CURNOTOPEN – попытка выполнить FETCH или CLOSE для курсора, который не был открыт;
- APPENDACTIVE – попытка выполнить start append, когда предыдущий start append не был завершен при помощи end append;
- APPENDNOTSTARTED – перед подачей putm не было вызвано оператора start append, или он завершился неудачно;

- QUERYWHENAPPEND – между start append и end append нельзя использовать execute direct;
- <число> - код завершения операции СУБД ЛИНТЕР;
- CUSTOM - пользовательское исключение.

Имена пользовательских исключений локальны в каждой процедуре (они возвращаются с помощью функции raise_error()).. При определении таких исключений им автоматически присваиваются внутренние коды. Чтобы передать такое исключение на вызывающий уровень (RESIGNAL) и корректно обработать его там, необходимо, чтобы коды этого исключения как на вызывающем, так и на уровне процедуры совпадали. Для этого можно явно задать код для пользовательского исключения (целое число):

Необработанное исключение в триггере эквивалентно FALSE (запрету). После исключения в одном из триггеров (если они могут запретить операцию) выполнение других триггеров продолжается. Т.е. все триггеры будут выполнены (однако, если один из BEFORE-триггеров запретил операцию, то операция не выполнится, и соответствующие AFTER- триггеры тоже не выполнятся).

CUSTOM <код>

Пользовательские исключения никогда не совпадают с кодами завершения СУБД ЛИНТЕР (они хранятся со знаком минус).

Пример блока описаний

```
Declare
  var a,b typeof(result);
  var i int default 0;
  exception badcur for badcursor;
  exception notab for 2202;
```

Формат блока кода

Блок кода хранимой процедуры имеет следующий формат:

CODE
<операторы>

Формат операторов рассматривается ниже.

Формат блока обработки исключений

Блок обработки исключений имеет следующий формат:

EXCEPTIONS
  WHEN <список имен исключений> **THEN**
    <операторы>**| IGNORE**
  [**WHEN** <список имен исключений> **THEN**
    <операторы>**| IGNORE**]
  …
  [**WHEN OTHERS THEN**
    <операторы>**| IGNORE**]
  [**WHEN ALL THEN**

<операторы>| **IGNORE**]

<Список имен исключений> – это одно или более определенных в блоке описаний имен, разделенных запятыми.

Фраза **WHEN** <список имен исключений> **THEN** <операторы> задает обработку конкретных декларированных в блоке описаний исключений. Исключения можно группировать в отдельные подгруппы в зависимости от алгоритма их обработки.

Фраза **WHEN OTHERS THEN** <операторы> задает обработку всех декларированных в блоке описаний исключений, обработка которых не предусмотрена в предыдущих фразах **WHEN**….

Фраза **WHEN ALL THEN** <операторы> гарантирует обработку всех возникших в процедуре исключений. Она относится ко всем не обрабатываемым в предыдущих фразах **WHEN„,** исключениям (как декларированным в блоке описаний, так и не декларированным),

Фраза **IGNORE** заставляет процедуру игнорировать перечисленные критичные исключения. Критичными считаются исключения: DIVZERO, UNDEFPROC, BADPARAM, BADRETVAL, BADCURSOR, CURNOTOPEN. Некритичные исключения игнорируются по умолчанию.

Когда в процедуре происходит указанное исключение, управление передается на соответствующие операторы после WHEN, которые выполняются до следующего WHEN, после чего происходит возврат из процедуры (аналогично оператору RETURN – см. стр. 25). Выполнение процедуры может быть продолжено с помощью оператора GOTO (см. стр. 24).

Если фразы WHEN, OTHERS и WHEN ALL не заданы, то исключения, для которых не задан код обработки, но которые определены в блоке описаний, автоматически передаются на верхний уровень (RESIGNAL). Все неопределенные в блоке описаний исключения, кроме критичных для исполнения, игнорируются. Критичными считаются следующие исключения: DIVZERO, UNDEFPROC, BADPARAM, BADRETVAL, BADCURSOR, CURNOTOPEN. Если происходит такое исключение, и для него нет обработчика, автоматически выполняется RESIGNAL.

Пример блока обработки исключений

```
Exceptions
  when notab then
    close first_cursor;
    goto notab_recovery;
  when badcur then
    success := false;
  when others then
    success := false;
    resignal;
```

Пример использования фразы WHEN ALL…

```
alter  procedure "te"(in i int) result int for debug
declare
  exception e1 for custom 1;
  exception e2 for custom 2;
code
  if i < 0 then
    signal e1;
  elseif i > 10 then
    signal e2;
  endif
  execute direct "select * from qwert;";
  return 0;
exceptions
  when e1 then
   return 1;
  when others then
   return 2;
  when all then
   return -1;
end;
```

Примеры использования фразы IGNORE

```
1)
create or replace procedure TEST (in i int) result int
declare
  exception DIVZERO for DIVZERO;
  var j int;
code
  j:=10/0;
  return 1;
  exceptions when DIVZERO then ignore;
end;

execute TEST(1);
Результат выполнения:
Return value = 1


2)
create or replace procedure TEST (in i int) result int
declare
  exception DIVZERO for DIVZERO;
  exception CURNOTOPEN for CURNOTOPEN;
  var j int;
  var c cursor(i int);
code
  close c;
  j:=10;
  return j;
  exceptions when DIVZERO then resignal; when others then ignore;
end;


3)
create or replace procedure TEST (in i int) result int
declare
  exception CURNOTOPEN for CURNOTOPEN;
  exception DIVZERO for DIVZERO;
  var j int;
  var c cursor(i int);
code
```

```
  close c;
  j:=10;
  return j;
  exceptions when all then ignore;
end;


4)
create or replace procedure TEST (in i int) result int
declare
  var j int;
  var c cursor(i int);
code
  close c;
  j:=10;
  return j;
  exceptions when all then ignore;
end;


5)
create or replace procedure TEST (in i int) result int
declare
  exception DIVZERO for DIVZERO;
  exception CURNOTOPEN for CURNOTOPEN;
  var j int;
  var c cursor(i int);
code
  close c;
  j:=10;
  return j;
  exceptions when DIVZERO, CURNOTOPEN then ignore; //
end;
```

Операторы

Общий вид любого исполняемого оператора следующий:

[<метка>:] <тело оператора>

<Метка> представляет собой имя, за которым следует двоеточие.

<тело оператора> определяется его функциональным назначением.

Оператор – выражение

Назначение

Назначением оператора является вычисление заданного выражения. Обычно смысл этого оператора заключается в достижении побочного эффекта при вычислении выражения, а именно присвоении значений и вызове стандартных функций. Синтаксис выражений в хранимых процедурах рассмотрен ниже.

Синтаксис

<выражение>;

Примеры

```
i := 0;
```

```
sum := eif[ c.i < 100 ] sum + 10 else sum + 100;
```

Условный оператор

Назначение

Условный оператор предназначен для организации ветвлений в процессе выполнения хранимой процедуры или триггера.

Формат оператора

IF <выражение условия 1> **THEN**
<операторы 1>
[**ELSEIF** <выражение условия 2> **THEN**
<операторы 2>
]
….
[**ELSE**
<операторы N>
]
ENDIF

Описание

Условный оператор может содержать нуль или несколько ветвей ELSEIF, а после них нуль или одну ветвь ELSE. При его исполнении вычисляется <выражение условия 1>, результат которого должен быть логического типа. Если его значение TRUE, выполняются <операторы 1>, и далее управление передается на следующий после ENDIF оператор. Если его значение FALSE, аналогичные действия выполняются с выражениями и операторами последующих ветвей ELSEIF, если они есть. Если ветвей ELSEIF нет или все значения их выражений FALSE, то выполняются <операторы N> ветви ELSE или управление передается за ENDIF, если ветви ELSE нет.

Пример условного оператора

```
If n > -1 and n < 0.5 then
  call processing(1);
elseif n >= 0.5 and n < 2 then
  call processing(2);
elseif n >= 2 and n < 10 then
  call processing(3);
Else
  call processing(4);
  Endif
```

Оператор выбора

Назначение

Оператор выбора предназначен для организации множественных ветвлений. Функционально его можно заменить условным оператором с соответствующим количеством условий ELSEIF, но он имеет более простую и наглядную конструкцию.

Формат оператора

CASE <выражение>
WHEN <константа1 [,константа]… [,константа]> **THEN**
<операторы 1>
WHEN <константа2 [,константа]… [,константа] > **THEN**

<операторы 2>
....
[**WHEN OTHERS THEN**
<операторы>
]
ENDCASE

Описание

Тип констант и тип CASE-выражения должны быть совместимы.

Значение CASE-выражения последовательно сравнивается с константами в операторах WHEN. При первом совпадении значения CASE-выражения с константой одного из WHEN-оператора выполняются операторы, соответствующие этому условию (т.е. до следующего WHEN-оператора или до ENDCASE), после чего выполнение оператора выбора завершается. Если значение CASE-выражения не было найдено среди списка констант, выполняется ветвь WHEN OTHERS. Если она отсутствует, управление передается сразу на следующий оператор после ENDCASE.

Пример оператора выбора

```
case rel_name
  when "$$$SYSRL", "$$$ATTRI", "$$$USR" then
    definit := "System dictionary table";
  when  "$$$PROC", "$$$PRCD" then
    definit := "System stored procedures table";
  when others then
    definit := "Unknown table class";
Endcase
```

Оператор цикла

Назначение

Оператор циклической обработки предназначен для организации многократного выполнения фрагментов программы.

Формат оператора

WHILE <выражение> **LOOP**
<повторяющиеся операторы>
ENDLOOP

Описание

WHILE-выражение должно быть логического типа. Если его значение TRUE, выполняются <повторяющиеся операторы> до ENDLOOP, после чего снова вычисляется WHILE-выражение, и далее все повторяется. Если значение WHILE-выражения FALSE, управление передается на следующий после ENDLOOP оператор. Чтобы количество циклов было конечным, в группе <повторяющиеся операторы> обязательно должно выполняться изменение переменных или условий, влияющих на WHILE-выражения.

Пример циклической обработки (классический перебор ответов в выборке)

```
while not outofcursor(curs) loop
  // обработка ответа
  sum := sum + curs.i + curs.j;
  execute direct "update tab1 set s = " + itoa(sum) +
                 "where current of \"CURS\";";
```

```
   fetch curs;
Endloop
```

Безусловный переход

Назначение

Оператор безусловного перехода предназначен для управления выполнением программы.

Формат оператора

GOTO <имя метки>;

Описание

В результате выполнения этого оператора управление передается на оператор, перед которым стоит указанная метка (имя метки пишется без двоеточия на конце).

Пример

```
goto tab_recovery;
```

Вызов хранимой процедуры

Формат вызова

CALL <имя> (<список параметров>) [**INTO** <имя переменной>];

Описание

Данный оператор исполняет другую хранимую процедуру или саму себя, в зависимости от имени. Любые виды рекурсивных вызовов разрешаются.

<Список параметров> – это список выражений или имен переменных, разделенных запятыми (может быть и пустым). Фактические параметры ставятся в соответствие формальным по порядку следования.

Если необходимо передать параметр по умолчанию, можно пропустить фактический параметр и сразу поставить запятую. Таким образом, чтобы пропустить несколько параметров, необходимо подряд поставить несколько запятых. Запятые не обязательно ставить в конце списка. Все недостающие параметры всегда получают свои значения по умолчанию, в том числе, если список пустой.

Если формальные параметры вызываемой процедуры имеют модификатор IN, то в качестве фактических параметров можно использовать любые выражения (совместимость типов выражений с типами параметров проверяется на этапе выполнения процедуры – см. далее).

Если формальные параметры вызываемой процедуры имеют модификатор OUT или INOUT, в качестве фактических параметров указывать выражения недопустимо. Здесь используются только имена переменных, в которые будут записаны выходные параметры. Если имя переменной опущено, выходное значение никуда записано не будет.

Если в операторе CALL задана фраза INTO, возвращаемое значение процедуры присвоится указанной переменной (такое использование недопустимо для процедур, возвращающих курсор; для передачи курсора в вызывающую процедуру существует специальная конструкция – см. ниже).

Разрешение ссылки на вызываемую процедуру происходит на этапе выполнения данной процедуры – процедура ищется по имени. Если такой процедуры нет, происходит исключение UNDEFPROC. Соответственно, на этапе выполнения так же проверяются типы и количество параметров и тип возвращаемого значения. В случае ошибок происходят исключения BADPARAM или BADRETVAL.

Пример оператора вызова

```
call myproc("auto",,1,aa) into bb;
```

Возврат из процедуры

Формат

RETURN [<значение>];

Описание

<Значение> – это некоторое выражение или имя курсорной переменной, если процедура возвращает курсор. Если значение не указано, процедура вернет значение NULL. В результате исполнения этого оператора выполнение процедуры завершается, и управление передается либо в вызывающую процедуру, если такая есть, либо формируется и отсылается ответ на запрос, вызвавший процедуру.

При возврате из процедур out-параметры и result-значения типа CHAR дополняются до заданной длины справа пробелами.

Пример оператора возврата

```
return sum * a;

create or replace PROCEDURE VCHAR1 (out OUTVAL varchar(12)) result varchar(12) for debug
declare
var a varchar(12);//
code
a := "abcd";//
OUTVAL:=a;//
return a;//
END;
```

Вызов исключения

Формат

SIGNAL <имя исключения>;

Описание

Этот оператор явно вызывает исключение с указанным именем (имя должно быть ранее определено в блоке описаний).

Пример

```
signal badparam; // сигнализировать о недопустимом значении
                 // входного параметра
```

Передача исключения

Формат

RESIGNAL;

Описание

Этот оператор можно использовать только в блоке обработки исключений. Его действием является завершение процедуры (со значением NULL в качестве возвращаемого значения) и передача возникшего исключения на верхний уровень (вызвавшую процедуру или ответ на запрос пользователя).

Обычно все произошедшие в процедуре исключения (кроме нескольких критичных, которые перечислены выше) не передаются на верхний уровень, если явно не вызывается RESIGNAL.

Открытие курсора

Оператор открытия курсора предназначен для связывания курсорной переменной с объектом процедурного языка, который будет заполнять поля курсора в процессе выполнения программы. Таким объектом может быть запрос на выборку данных или процедура, возвращающая курсор.

Формат оператора

OPEN <имя курсорной переменной> [**AS** “имя курсора”] **FOR** <запрос>;

или

OPEN <имя курсорной переменной> [**AS** “имя курсора”] **FOR**
CALL <имя процедуры>(<список параметров>);

Описание

Курсор представляет собой выборку, содержащую набор значений определенной структуры (состоящую из полей). Значения полей доступны через курсорную переменную, а навигация по выборке осуществляется оператором FETCH (см. стр. 27).

Формат <запроса> см. на стр. 29. Для открытия курсора допустим только SELECT - запрос.

Во втором случае предполагается вызов процедуры, возвращающей курсор. Синтаксис вызова такой же, как в операторе CALL, за исключением того, что здесь нельзя использовать фразу INTO.

Если указана фраза AS, открывается поименованный курсор. Курсор необходимо именовать в случае, когда будут подаваться запросы с условием WHERE CURRENT. Имя курсора заключается в кавычки и должно быть допустимо с точки зрения СУБД ЛИНТЕР.

После открытия курсора значения первого ответа выборки уже доступны через курсорную переменную (см. стр. 27).

Если количество и/или типы полей курсора не соответствуют структуре курсорной переменной, при выполнении OPEN происходит исключение BADCURSOR.

Примеры

```
open a for direct "select * from auto where make = '" +
                  make + "';" ;

open b as "cursor_b" for call retcur("auto");
```

Выборка данных из открытого курсора

Формат

FETCH <имя курсорной переменной> [<ориентация>] [**INTO** <переменная> [, …]];

Описание

Данный оператор выбирает очередной ответ из курсора согласно ориентации. Результаты ответа доступны через переменные, заданные в опции INTO или через поля курсорной переменной, обращение к которым осуществляется следующим образом:

<имя курсорной переменной>.<имя поля>

<Ориентация> задается одним из следующих способов:

- NEXT – следующий ответ;
- PREVIOUS – предыдущий;
- FIRST – первый;
- LAST – последний;
- ABSOLUTE <выражение> – ответ с номером, вычисленным в выражении (типа INTEGER или SMALLINT);
- RELATIVE <выражение> – ответ, расположенный на вычисленном в выражении расстоянии от текущего ответа (расстояние должно быть целочисленным).

Если ориентация не задана, предполагается NEXT.

Для организации цикла по выборке используется комбинация операторов FETCH и обычного цикла WHILE, в условии которого стоит вызов стандартной функции outofcursor (см. стр. 89). Кроме outofcursor есть еще функции rowcount для получения количества ответов и errcode для получения кода ошибки (в случае, если эта ошибка не перехватывается исключением).

Описание конструкции INTO <переменная> [, …] приведено в разделе «Выполнение некурсорных запросов».

При попытке сделать FETCH для неоткрытого курсора происходит исключение CURNOTOPEN.

Примеры

```
fetch a;

fetch b previous;

fetch c relative offset * size - 1;
```

Закрытие курсора

Назначение

Оператор закрытия курсора предназначен для прекращения работы с открытым курсором и освобождения используемых при организации курсора ресурсов.

Формат оператора

CLOSE <имя курсорной переменной>;

Описание

Использование оператора CLOSE не обязательно: при завершении программы все открытые курсоры закрываются автоматически, кроме курсора, который возвращается как результат в процедуре курсорного типа. В последнем случае курсор, наоборот, **не должен** явно закрываться.

При попытке закрыть неоткрытый курсор происходит исключение CURNOTOPEN.

Выполнение некурсорного запроса

Назначение

Оператор некурсорного запроса предназначен для организации непосредственной работы с базой данных.

Формат оператора

EXECUTE <запрос>[**INTO** <переменная> [, …]];

Механизм курсоров позволяет использовать только SELECT-запросы при работе с базой данных. Оператор EXECUTE позволяет выполнить любой запрос (в том числе и SELECT) по отдельному каналу СУБД ЛИНТЕР, который неявно открывается при обработке этого оператора (его закрытие производится в зависимости от обработки транзакции).

Параметр INTO <переменная> [, …] задает список скалярных переменных, в который должны быть загружены выбираемые по <запросу> значения. Количество и тип данных скалярных переменных должны соответствовать количеству и типу данных выбираемых по <запросу> значений.

Запрос выборки должен возвращать только одну запись.

В результате выполнения оператора EXECUTE может возникнуть исключение, соответствующее коду завершения СУБД ЛИНТЕР (так же, как и при выполнении операторов OPEN, FETCH и CLOSE). Синтаксис запросов рассматривается в следующем параграфе.

Для некурсорных запросов рекомендуется использовать собственный обработчик исключений.

Примеры

```
1)
execute direct "update tab1 set s = " + itoa(sum) +
                     "where current of \"CURS\";";

2)
execute direct "create table test(i int);";

3)
var cnt int; //
…
execute direct "select count(*) from auto;" into cnt; //

4)
create or replace procedure tst() result  char(50) for debug
```

```
Declare
var mdl char(20); //
var sale int; //
Code
execute "select model, year+1900  from auto where personid=500;" into mdl,sale; //
Return mdl+" дата продажи; "+ itoa(sale); //
end;
```

Завершение транзакции

Назначение

Операторы завершения транзакции предназначены для подтверждения или отказа от внесенных в базу данных изменений в процессе выполнения текущей транзакции.

Формат операторов

COMMIT [**RELEASE**]; // фиксация изменений
ROLLBACK [**RELEASE**]; // откат

Описание

Все изменения в базу данных вносятся с помощью запросов, подаваемых в операторе EXECUTE. Все запросы оператора EXECUTE в хранимой процедуре подаются по каналу, который автоматически открывается как дочерний от пользовательского канала в приложении (то есть канала, по которому исполняется пользовательская программа). Процедура может откатить или зафиксировать те изменения, которые были сделаны в ее теле, либо просто завершить исполнение. В последнем случае решение вопроса о результате завершения транзакции полностью возлагается на приложение, которое могут подать операторы COMMIT или ROLLBACK по своему каналу, что отразится на всех изменениях, сделанных как самой процедурой, так и всеми ее дочерними процедурами.

Таким образом, процедура может частично откатывать или фиксировать свои изменения, или использоваться как часть более крупной транзакции.

Если в этих операторах указана фраза RELEASE, внутренний курсор, по которому выполняются операторы EXECUTE, сразу закрывается. В противном случае он остается открытым, что позволяет быстро продолжить сеанс изменений.

Запросы

В хранимых процедурах допустимы два вида запросов: *претранслируемые*, которые разбираются на этапе трансляции процедуры, и *динамические*, которые формируются на этапе выполнения процедуры и сразу же выполняются. Приоритет обеих типов запросов наследуется от родительского канала.

Претранслируемые запросы

Формат претранслируемого запроса

<текст запроса> [,<список параметров>]

Описание

- <текст запроса> – символьная константа (то есть строка в кавычках);
- <список параметров> – список выражений, разделенных запятой;
- текст запроса с параметрами должен быть на момент трансляции процедуры. Запрос транслируется и сохраняется в коде процедуры. На этапе выполнения к претранслированному запросу привязываются вычисленные параметры, если они есть.

Примеры

```
1)
create or replace procedure tst(in id int) result cursor( c char(20) )
   declare
   var b typeof(result);
   code
   open b for "select make from auto where personid=?;",id;
   return b;
   end;
```

2)
```
create or replace procedure tst(in id int) result cursor( c char(20) )
   declare
   var b typeof(result);
   code
   open b for "select make from auto where personid=:param;",id;
   return b;
   end;
```

Динамические запросы

Формат динамического запроса

DIRECT <выражение символьного типа>[**INTO** <переменная> [, …]];

Описание

При исполнении запроса этого вида вычисляется соответствующее символьное выражение, которое трактуется как запрос на языке SQL.

Если при выполнении запроса обнаруживается ошибка, происходит исключение с соответствующим кодом.

Описание конструкции INTO <переменная> [, …] приведено в разделе «Выполнение некурсорных запросов».

Выражения

Выражение – это комбинация объектов процедурного языка, называемых *операндами* выражения. В качестве операндов могут выступать локальные переменные, значения переданных параметров, отдельные поля структуры константы, стандартные функции или другие, более простые выражения. Операнды объединяются в выражение при помощи арифметических и логических операторов, а также операторов отношения.

Действия над операндами при вычислении значения выражения выполняются согласно приоритету операций.

Порядок выполнения операций внутри выражения может быть изменен при помощи круглых скобок.

Для наглядности любое выражение может быть заключено в фигурные скобки (символы «{ }») и так включаться в другие выражения. Последовательность выражений, разделенных точкой с запятой (символ «;»), также может рассматриваться как выражение. В этой последовательности значения всех подвыражений вычисляются последовательно слева направо, а результатом всего выражения считается результат последнего подвыражения.

Имеется специальный синтаксис условного выражения (не путать с условным оператором!), значением которого может быть значение одного из двух подвыражений в зависимости от значения третьего подвыражения логического типа (условия).

Выражения различаются по типам, в зависимости от типа результата. Типы выражений соответствуют типам, обрабатываемым в хранимых процедурах. Исключением являются курсоры. Разрешается лишь присвоение одной курсорной переменной другой.

По умолчанию целочисленные константы имеют тип SMALLINT, поэтому при использовании их в выражениях возможно переполнение результата выражения (например, такого, как 2*2*2*2*2*2*2*2*2*2*2*2*2*2*2*2). Чтобы результат выражения формировался правильно, необходимо явно указать тип константы при помощи суффикса (достаточно для одного операнда выражения). Тип результата будет всегда приводиться к большей размерности (например, 2*2i*2*2, 2*2i*2*2b).

В текущей версии СУБД ЛИНТЕР присвоение курсорных переменных не реализовано и не дает никакого эффекта.

Если при вычислении выражения происходит попытка деления на ноль, генерируется исключение DIVZERO, а результатом деления считается 0.

Максимальная длина результата символьного или байтового выражения – 3919 символов (байтов) (для выражений типа UNICODE- 1959 байтов).

Операнды

Для включения в выражение локальных переменных, параметров и функций используются их имена.

Для именования поля курсора используется следующая конструкция:

```
<имя курсора>.<имя поля>
```

Для вызова стандартной функции используется следующая конструкция:

```
<имя функции>([<список параметров>])
```

Список параметров может быть пуст или состоять из одного либо нескольких выражений, разделенных запятыми. Параметры передаются в функцию по порядку следования. Несколько последних параметров могут быть опущены , они получат значения по умолчанию. Однако пропускать параметр в середине при вызове стандартных функций нельзя. Параметры стандартных функций только входные.

Соответствие типов параметров при вызове стандартных функций проверяется на этапе трансляции. На этапе выполнения проверяется допустимость значений параметров: если

в стандартную функцию передается недопустимое значение, вызывается исключение BADPARAM.

Любое выражение может рассматриваться как операнд, если оно при помощи некоторой операции включается в более сложное выражение.

Операции

Ниже рассматриваются все операции по типам выражений в порядке убывания их приоритетов.

Операции в числовых выражениях

Наивысший приоритет имеют унарные операции «–» или «+».

Операция «–», стоящая перед числовым операндом, рассматривается как унарный минус, то есть изменение знака числа.

Операция «+» – унарный плюс, не влияет на значение числа.

Второй уровень приоритета имеют бинарные операции:

- «*» – умножение;
- «/» – деление;
- «\» – деление с отбрасыванием дробной части частного (деление нацело);
- «^» – возведение левого операнда в степень правого.

Если при выполнении операций деления значение делителя равно 0, происходит исключение DIVZERO, а результатом деления считается 0.

Низший уровень приоритета имеют бинарные операции:

- «+» – сложение;
- «–» – вычитание.

Операции в символьных выражениях

В символьных выражениях допустимы только две операции, имеющие одинаковый приоритет:

- «+» – конкатенация (слияние) строк;
- «–» – конкатенация строк без промежуточных пробелов.

Максимальное число строк-операндов в операции конкатенации равно 255.

Операции в логических выражениях

Наивысший приоритет среди логических операций имеют операция NOT (логическое отрицание), затем операции отношения.

Операция NOT унарная, ее формат:

NOT<логическое выражение>

Существуют следующие операции отношения:

- = – проверка на равенство;
- <> – проверка на неравенство;
- > – проверка на больше;
- < – проверка на меньше.
- >= – проверка на больше или равенство;
- <= – проверка на меньше или равенство.

Все операции отношения бинарные, они должны соединять два сравниваемых выражения совместимого типа. Допускается сравнение выражений всех типов, кроме курсорных переменных. При этом выражения логического типа могут сравниваться только ограниченно: на равенство или неравенство. Выражение любого типа можно так же сравнить с NULL на равенство или неравенство. Сравнение дат производится точно, с учетом времени до тиков.

Операция AND – логическое «И», бинарная, должна связывать два логических выражения. При вычислениях второе выражение вычисляется только в случае истинности первого.

Операция OR – логическое «ИЛИ», бинарная, должна связывать два логических выражения. При вычислениях второе выражение вычисляется только в случае ложности первого.

Операции в выражениях типа «дата»

С операндами (или выражениями) типа «дата» (типа DATE) допустимы следующие операции:

- «$» – вычисление разности между двумя датами в днях (результат – целое число). Оба операнда должны иметь тип данных DATE. При вычислении разности дат часы не учитываются. Если первая дата меньше второй, то разность будет представлена отрицательным числом. Эта операция имеет наивысший приоритет;
- «+» – прибавление к дате определенного количества дней;
- «–» – вычитание из даты определенного количества дней.

В операциях прибавления (вычитания) дней к дате выражение слева должно иметь тип DATE, а выражение справа должно быть числового типа.

Присвоение значений

Присвоение значений объектам процедурного языка может быть выполнено различными способами в зависимости от типа объекта:

- при объявлении локальных переменных с помощью параметра значение по умолчанию DEFAULT;
- при выполнении операции присвоения;
- значение входным параметрам функций и процедур присваивается автоматически при их вызове;
- значение полям курсора может быть присвоено либо при выполнении FETCH, либо оператором присвоения;

- значение предопределенным переменным в триггерах присваивается СУБД автоматически.

Назначение

Операция присвоения предназначена для присвоения переменным новых значений.

Формат операции присвоения

<имя переменной>:=<выражение>

Описание

При выполнении присвоения сначала вычисляется значение <выражения>, затем полученное значение присваивается указанной переменной.

Присвоение рассматривается как выражение, значением которого является значение <выражения> справа, поэтому его можно использовать как часть более сложного выражения (аналогично языку программирования C/C++). В частности, допустима запись типа `a := b := c := 0`.

В качестве правой части присвоения может использоваться NULL. Такое присвоение тоже интерпретируется как NULL, поэтому допустимо присвоение вида
`i := s := d := NULL`
даже если `i`, `s` и `d` имеют разные типы (NULL совместим со всеми типами).

Условные выражения

Формат выражения

EIF’[‘<логическое выражение>‘]’<выражение1>[**ELSE** <выражение2>]

Описание

Здесь квадратные скобки после EIF являются обязательной частью синтаксиса.

При вычислении условного выражения сначала вычисляется <логическое выражение>, затем, если его результат TRUE, вычисляется <выражение1>, иначе вычисляется <выражение2>, если оно указано.

В условном выражении ветвь ELSE <выражение2> может отсутствовать. В этом случае при ложном значении <логического выражения> результат условного выражения - NULL.

Пакетное добавление

Возможны два режима добавления строк в таблицу: обычный и пакетный.

В обычном режиме добавление строки выполняется с помощью оператора INSERT. Строки добавляются по одной при каждом исполнении данного оператора.

В пакетном режиме строки добавляются за одну операцию в том количестве, в каком они заданы во входном буфере. Пакетный режим рекомендуется использовать при загрузке больших объемов данных.

Для выполнения пакетного добавления внутри хранимой процедуры:

1. определить курсор, в котором имена полей соответствуют именам (и типам) полей загружаемой таблицы. Если имя поля таблицы содержит нестандартные символы, совпадает с ключевым словом и т.д. (т.е. требует квотации), в описании поля курсора можно явно указать имя, какое должно использоваться. Для этого после типа столбца можно указать ключевое слово column и строку в кавычках, например:

```
cursor(
i int; // будет соответствовать колонке "I"
l char(20) column "lowercase" // будет
//соответствовать колонке "lowercase"
ins date column "INSERT" // будет соответствовать
//колонке "INSERT"
)
```

2. перед добавлением выдать оператор:

```
start append into <таблица> from <курсор>;
```

   где <таблица> – имя таблицы;

   <курсор> – курсорная переменная.

   Выполнение этого оператора внутри процедуры приводит к подаче SQL-запроса start append into byte (<список столбцов> согласно именам полей в таблице).

   В случае ошибки при выполнении оператора формируется исключение с кодом ошибки.

   В случае, если <курсор> – недопустимое выражение, вызывается исключение BADPARAM.

   При попытке выполнить start append, когда предыдущий start append не был завершен при помощи end append, возникает исключение APPENDACTIVE.

3. чтобы добавить строку, надо заполнить поля соответствующими значениями и выдать оператор:

```
putm <буфер>;
```

   В качестве <буфера> можно использовать любую переменную типа курсор, структура которой совпадает со структурой переменной, использованной в операторе start append. В случае несовпадения возникнет исключение BADCURSOR.

   Этот оператор накапливает данные во внутренней странице пакета (которая выделяется оператором start append) и, если пакет заполняется, загружает его в таблицу.

   Если в процессе формирования пакета и загрузки его в таблицу возникает ошибка, то вырабатывается исключение с кодом завершения, который можно получить с помощью функции errcode().

   Если возникает ошибка, внутренняя страница продолжает содержать данные, которые не удалось добавить, а новая запись игнорируется.

   Чтобы очистить внутреннюю страницу, можно использовать функцию

```
clearPutm ()
```

   Узнать количество записей во внутренней странице (которые еще не занесены реально в БД) можно при помощи функции

```
int getPutmRecs ()
```

Эту функцию можно использовать, в частности, после возникновения ошибки в putm. Ошибка означает, что не все записи из внутренней страницы добавлены. Функция

```
getPutmRecs ()
```

позволяет узнать, сколько именно записей не добавлено, чтобы попытаться добавить эти последние n записей по одной, проверяя, какая именно из записей пакета ошибочна.

Можно потребовать принудительно сбросить записи из внутренней страницы в БД при помощи функции

```
flushPutm ()
```

В случае успеха внутренняя страница освобождается для последующей нормальной работы.

В случае ошибки обработка внутренней страницы аналогична выполнению функции putm, попытавшейся сбросить заполненную внутреннюю страницу и столкнувшейся с ошибкой: внутренняя страница не очищается. Очистить ее всегда можно при помощи clearPutm().

Таким образом, вызов подряд putm и flushPutm() приведет, фактически, к добавлению строк по одной через механизм пакетного добавления.

Если перед подачей putm вызовом функций getPutmRecs, clearPutm и flushPutm не был выполнен оператор start append, или он завершился неудачно, то при попытке выполнить putm возникнет исключение APPENDNOTSTARTED.

4. по окончании добавления надо выдать оператор end append.

   Если перед подачей end append не было вызвано оператора start append, или он завершился неудачно, то при попытке выполнить end append возникнет исключение APPENDNOTSTARTED.

   После каждого start append должен вызываться end append, прежде чем делать другие execute direct или start append!

5. все операции putm выполняются по одному каналу, по тому же, что и execute direct в процедурах.

   Соответственно, между start append и end append нельзя использовать другие execute direct. Если пытаться сделать это, возникнет исключение QUERYWHENAPPEND (если его не обработать, последующая попытка выполнения SQL-оператора приведет к ошибке 1013 – неверная последовательность команд).

Пример

```
alter procedure "PM"() result char(20) for debug
declare
   var c cursor(i int column "start",
                si smallint, bi bigint,
                c char(20), vc varchar(30),
                d date column "THIS IS DATE",
                r real, db double, dc numeric,
                l bool,
                b byte(10), vb varbyte(10)
   );
   var i int;
```

```
    exception APPENDNOTSTARTED for APPENDNOTSTARTED;
    exception QUERYWHENAPPEND for QUERYWHENAPPEND;
    exception APPENDACTIVE for APPENDACTIVE;
    exception AAA for 116;
  Code
    execute direct "drop table pm;";
    execute direct "create table pm("
                   "\"start\" int, si smallint, bi bigint,"
                   "c char(20), vc varchar(30), "
                   "\"THIS IS DATE\" date,"
                   "r real, db double, dc decimal,"
                   "l boolean,"
                   "b byte(10), vb varbyte(10));";
    start append into "PM" from c;
    i := 1;
    while i < 32001 loop
      c.i := i;
      c.si := i;
      c.bi := i*100;
      c.c := "string value "+tochar(i);
      c.vc := c.c + " ";
      c.d := sysdate() + i;
      c.r := i/10.0;
      c.db := i/100.0;
      c.dc := i/1000.0;
      c.l := mod(i, 10) = 0;
      asc(tochar(i), c.vb);
      asc(tochar(sysdate()+i), c.b);
      putm c;
      if errcode() <> 0 then
        Return "PUTM error: "+tochar(errcode());
      endif
      i := i+1;
    endloop
    end append;
    if errcode() <> 0 then
      return "Error "+tochar(errcode());
    endif
    return "Ok";
  Exceptions
    when AAA then
      return "PUTM failed";
    when APPENDNOTSTARTED then
      return "APPENDNOTSTARTED";
    when APPENDACTIVE then
      return "APPENDACTIVE";
    when QUERYWHENAPPEND then
      return "QUERYWHENAPPEND";
  end;
```

При сравнении этой процедуры с процедурой, которая добавляет те же строки не в пакетном режиме, а по одной (с помощью INSERT), были получены следующие характеристики.

```
Время работы:
10 000 строк:
insert: 33 с.
putm: 11 с.
32 000 строк:
insert: 2 мин.
```

```
putm: 36 с.
```

Если добавлять меньше столбцов, выигрыш от putm более очевидный.

```
Для 4 столбцов и 10 000 строк:
insert: 26 с.
putm: 5 с.
```

Работа с типом данных BLOB

В процедурном языке работа с типом данных BLOB реализована с помощью нескольких функций:

Имя	Функция
add_blob	Добавление значения в конец BLOB (запись в произвольное место не допустима)
read_blob read_blob_int read_blob_smallint read_blob_bigint read_blob_real read_blob_double read_blob_numeric read_blob_char read_blob_date read_blob_bool read_blob_nchar	Чтение значений из BLOB
seek_blob	
clear_blob	Удаление BLOB
blob_size	Получение информации из заголовка BLOB
set_cur_blob	Установки текущего BLOB-столбца при обработке запроса с несколькими BLOB-значениями

В момент операции с BLOB-значением должны быть определены текущая запись таблицы и номер BLOB-столбца, для которых будет производиться эта операция. Текущая запись устанавливается в следующих случаях:

- после успешного выполнения SELECT - запроса;
- после выборки определенного ответа (FETCH);
- после добавления новой записи в таблицу.

Номер BLOB-столбца, к которому добавляется порция данных, предварительно устанавливается с помощью функции set_cur_blob. Если это не сделано по умолчанию, используется первый BLOB-столбец.

Так как процедурный язык обеспечивает два способа выполнения манипуляций с БД, а именно, с помощью курсоров и оператора EXECUTE, в функциях работы с BLOB существует, соответственно, два способа указывать выборку, для которой производится операция с BLOB-значением:

- для курсоров в качестве первого параметра функции указывается курсорная переменная и далее остальные параметры;

- если необходимо работать с BLOB-значением для записи, которая была выбрана в результате выполнения SELECT-или INSERT-запроса оператором EXECUTE, ссылка на курсор не задается, и сразу перечисляются остальные параметры функции.

При работе с BLOB-данными первым способом (через курсор) в описании курсора используется ключевое слово BLOB для указания поля, которое в выборке соответствует колонке типа BLOB. При выборке ответов на SELECT-запрос для поля типа BLOB выбирается только информационный заголовок длиной 14 байт. Для работы со значениями BLOB (чтение, добавление, очистка) используются специальные стандартные функции (см. ниже). Фактически ключевое слово BLOB является синонимом для BYTE(14), в такое поле как раз и считывается заголовок. Полезной информацией, которую можно извлечь из заголовка, является размер порции данных BLOB. Для его получения используется функция blob_size.

Чтение данных из BLOB-значения организовано так, что каждый новый вызов функции чтения выбирает очередную информацию, расположенную в значении BLOB сразу после тех данных, которые были прочитаны предыдущим вызовом (или с начала, если с момента выборки текущей записи операций чтения еще не было). Это аналогично чтению данных из последовательного файла и достаточно удобно в использовании. Если необходимо произвести чтение с конкретной позиции в BLOB - значении, перед вызовом функции чтения можно установить указатель текущей позиции с помощью функции seek_blob.

Попытка работы с BLOB-значениями таким образом, когда сначала выбираем BLOB-значение, затем работаем с другой таблицей (например, делаем insert), затем снова возвращаемся к старому BLOB-значению в рамках внутреннего канала процедуры (то есть делаем это все не по разным курсорам), является недопустимой.

Чтобы реализовать такую схему работы, необходимо использовать дополнительный курсор, по которому ведется работа с BLOB-значениями.

Добавление значения в конец BLOB

Синтаксис

add_blob ([<курсор>,] <значение> [, <размер>])

<курсор> – курсорная переменная;
<значение> – переменная типа BYTE или значение любого другого скалярного типа;
<размер> – числовое выражение.

Описание

Функция добавляет порцию данных в конец установленного BLOB-столбца текущей записи.

Параметр <значение> определяет, что заносится в BLOB.

Возможны два случая:

1) если это переменная типа BYTE, в качестве порции данных берется соответствующая последовательность байт, указанная в параметре <размер> длины. Это самый общий способ, так как он позволяет записывать любые данные.
2) если <значение> – это выражение других типов, в BLOB записывается двоичное представление данного значения для конкретной машины. В частности, значения типа DATE записываются как два четырехбайтовых числа (типа INT), первое из

которых содержит количество дней, прошедших с начала нашей эры, а второе - количество тиков (сотых долей секунды), прошедших с начала дня (именно так представляются значения типа DATE внутри подсистемы хранимых процедур). Значения типа CHAR записываются как два байта длины строки и соответствующее количество байт самой строки после них. В этом случае длину порции данных указывать не требуется, так как она известна.

В обоих случаях, если длина переменной <значение> больше длины, указанной в параметре <размер>, то добавляемая порция данных будет обрезана до заданной в параметре <размер> длины; если меньше – дополнена нулями.

Все записанные в BLOB значения могут быть адекватно считаны с помощью функций чтения `read_blob`.

Возвращаемое значение

Функция возвращает код завершения СУБД ЛИНТЕР.

Чтение значения из BLOB на общем уровне

Синтаксис

read_blob ([<курсор>,] <буфер> , <размер>)

<курсор> – курсорная переменная;

<буфер> – переменная типа BYTE или значение любого другого скалярного типа;

<размер> – числовое выражение.

Описание

Функция выполняет чтение порции данных из установленного BLOB-столбца текущей записи на общем уровне, то есть последовательность байт из BLOB-значения заносится «как есть» в массив байт <буфер>.

Параметр <размер> определяет максимальный размер считываемой порции данных. Из BLOB считывается порция данных, меньшая или равная по длине значению параметра <размер> (размер порции меньше этого параметра, если в BLOB больше нет данных).

Возвращаемое значение

1) количество реально загруженных в <буфер> байт;
2) код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции `errcode()`.

Чтение int-значения из BLOB

Синтаксис

read_blob_int ([<курсор>])

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из установленного BLOB-столбца текущей записи значения типа `int`.

Предполагается, что в BLOB в текущей позиции для чтения находится двоичное представление значения `int` типа, как описано в функции `add_blob`.

Возвращаемое значение

1) считанное int-значение;
2) код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции `errcode()`.

Чтение smallint-значения из BLOB

Синтаксис

read_blob_smallint ([<курсор>])

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из установленного BLOB-столбца текущей записи значения типа `smallint`.

Предполагается, что в BLOB в текущей позиции для чтения находится двоичное представление значения `smallint` типа, как описано в функции `add_blob`.

Возвращаемое значение

1) считанное значение типа `smallint`;
2) код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции `errcode()`.

Чтение bigint-значения из BLOB

Синтаксис

read_blob_bigint ([<курсор>])

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из установленного BLOB-столбца текущей записи значения типа `bigint`.

Предполагается, что в BLOB в текущей позиции для чтения находится двоичное представление значения `bigint` типа, как описано в функции `add_blob`.

Возвращаемое значение

1) считанное значение `bigint`-типа;
2) Код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции `errcode()`.

Чтение real-значения из BLOB

Синтаксис

read_blob_real ([<курсор>])

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из установленного BLOB-столбца текущей записи значения типа real.

Предполагается, что в BLOB в текущей позиции для чтения находится двоичное представление значения real типа, как описано в функции add_blob.

Возвращаемое значение

1) считанное real-значение;
2) Код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Чтение numeric-значения из BLOB

Синтаксис

read_blob_numeric ([<курсор>])

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из установленного BLOB-столбца текущей записи значения типа numeric.

Предполагается, что в BLOB в текущей позиции для чтения находится двоичное представление значения numeric типа, как описано в функции add_blob.

Возвращаемое значение

1) считанное numeric-значение;
2) код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Чтение double-значения из BLOB

Синтаксис

read_blob_double ([<курсор>])

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из установленного BLOB-столбца текущей записи значения типа double.

Предполагается, что в BLOB в текущей позиции для чтения находится двоичное представление значения double типа, как описано в функции add_blob.

Возвращаемое значение

1) считанное double-значение;
2) код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Чтение char-значения из BLOB

Синтаксис

read_blob_char ([<курсор>])

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из установленного BLOB-столбца текущей записи значения типа char.

Предполагается, что в BLOB в текущей позиции для чтения находится двоичное представление значения char типа, как описано в функции add_blob.

Возвращаемое значение

1) считанное char-значение;
2) код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Чтение nchar-значения из BLOB

Синтаксис

read_blob_ncar ([<курсор>])

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из установленного BLOB-столбца текущей записи значения типа nchar.

Предполагается, что в BLOB в текущей позиции для чтения находится двоичное представление значения nchar типа, как описано в функции add_blob.

Возвращаемое значение

1) считанное nchar-значение;
2) код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Чтение date-значения из BLOB

Синтаксис

read_blob_date ([<курсор>])

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из установленного BLOB-столбца текущей записи значения типа date.

Предполагается, что в BLOB в текущей позиции для чтения находится двоичное представление значения date типа, как описано в функции add_blob.

Возвращаемое значение

1) считанное значение типа date;
2) код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Чтение bool-значения из BLOB

Синтаксис

read_blob_bool ([<курсор>])

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из установленного BLOB-столбца текущей записи значения типа bool.

Предполагается, что в BLOB в текущей позиции для чтения находится двоичное представление значения bool типа, как описано в функции add_blob.

Возвращаемое значение

1) считанное bool-значение;
2) код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Установка текущей позиции для чтения BLOB

Синтаксис

seek_blob ([<курсор>,] <позиция>)

<курсор> – курсорная переменная;

<позиция> – числовое выражение.

Описание

Функция задает позицию для чтения данных из установленного BLOB–столбца текущей записи, т.е. изменяет внутреннюю переменную в памяти исполняющей подсистемы и всегда выполняется успешно. Попытка установить указатель текущей позиции в недопустимое место будет обнаружена при последующей операции чтения.

Возвращаемое значение

Значение параметра <позиция>.

Получение размера BLOB-данных

Синтаксис

blob_size (<заголовок>)

<заголовок> – переменная типа BLOB или BYTE(14).

Описание

Функция возвращает размер в байтах BLOB–значения установленного столбца текущей записи, который считывается в поле курсора при выборке ответа, содержащего BLOB-значение.

Установка текущего BLOB-столбца

Синтаксис

set_cur_blob ([<курсор>],<номер>)

<курсор> – курсорная переменная, для которой устанавливается номер BLOB-столбца (если не задан, то функция применяется для канала оператора EXECUTE).

<номер> – номер BLOB-столбца.

Описание

Выполняет переключение всех функций по работе с BLOB-значениями на работу с заданным столбцом (по умолчанию – первый столбец типа BLOB). Кроме того, после ее вызова сбрасывается внутренний счетчик позиции для чтения, т.е. функции типа read_blob начнут читать значение с начала (требуется использовать seek_blob для установки на нужное место).

Типизация BLOB-столбца

Синтаксис

modify_blob_type ([<курсор>,]<тип BLOB-значения> [,<номер>])

<курсор> – курсорная переменная, для которой устанавливается номер BLOB-столбца (если не задан, то функция применяется для канала оператора EXECUTE).

<тип BLOB-значения> – переменная типа LONG, значение которой используется для идентификации данных BLOB-столбца (текст, графика, анимация, музыка и т. п.).

<номер> – порядковый номер BLOB-столбца.

Описание

Функция изменяет тип BLOB-значения заданного столбца в выборке. Типизация BLOB-значений регламентируется пользователем. Если аргумент *<номер>* не задан, по умолчанию используется первый BLOB-столбец выборки.

Пример

Создается таблица tblob, в нее заносятся 10 записей, затем устанавливаются типы BLOB-значений. В конце select-запросом проверяются установленные типы BLOB-значений.

```
create or replace procedure blobtest(in n int default 10)
declare
    var s cursor(i int, b1 blob, b2 blob); //
    var i,j int default 1; //
    var b byte(100); //
    exception addbloberr for custom 1; //
code
    execute direct "create or replace table tblob(i int, b1 blob, b2 blob);"; //
    while i <= n loop
```

```
        execute direct "insert into tblob(i,b1,b2) values(" + itoa(i) + ", NULL,
NULL);"; //
        j := 0; //
        while j < i * 3 loop
          b[j] := i * 10 + j; //
          j := j + 1; //
        endloop
        if add_blob(b, i * 3, 2) <> 0 then
           signal addbloberr; //
        endif
        if add_blob(b, i * 3, 3) <> 0 then
           signal addbloberr; //
        endif
        modify_blob_type(i + 100, 2); //
        i := i + 1; //
    endloop
    open s for direct "select i, b1, b2 from tblob;";  //
    while not outofcursor(s) loop
        modify_blob_type(s, s.i + 1000, 3); //
        fetch s;  //
    endloop
exceptions
when others then
    resignal; //
end;

execute blobtest();
select i, getlong(b1, 20), getlong(b2, 20) from tblob;
```

Поддержка кодовых страниц

В СУБД ЛИНТЕР каждое значение типа CHAR во внутреннем представлении снабжается информацией о его кодировке. Это позволяет прозрачным для пользователя способом организовывать работу с различными кодировками.

Для понимания использования различных кодировок необходимо представлять источники, из которых определяется кодировка той или иной строки:

1) все символьные константы внутри исходного текста процедуры (триггера) имеют кодировку, в которой работал клиент на момент создания этой процедуры (триггера);
2) при передаче символьных параметров в хранимую процедуру информация о кодировке извлекается из самого параметра, т.е. кодировка соответствует текущей рабочей кодировке клиента, подавшего запрос EXECUTE на исполнение процедуры, либо кодировке строки, переданной в данную процедуру из другой процедуры;
3) при вызове триггера кодировка символьных данных в переменных OLD и NEW соответствует текущей рабочей кодировке канала, по которому исполняется вызвавший триггер запрос;
4) при извлечении данных курсором кодировка символьных полей соответствует текущей рабочей кодировке канала, по которому работает открывшая курсор хранимая процедура (триггер);
5) при присвоении символьного значения переменной, данное значение сохраняется в переменной вместе с информацией о его кодировке (т.е. значение записывается без каких бы то ни было преобразований);

6) в бинарных операциях (например, конкатенация или сравнение строк) кодировка правого операнда преобразуется к кодировке левого операнда, и результат будет сохранен в кодировке левого операнда;
7) при выполнении запросов из хранимых процедур (триггеров) кодировка строки преобразуется к кодировке канала, по которому работает открывшая курсор хранимая процедура (триггер).

Надо помнить, что не всегда символы некоторой кодовой страницы могут быть преобразованы к символам другой кодовой страницы. Иногда преобразование может происходить с потерями (например, недопустимые символы могут заменяться знаком вопроса «?»).

Стандартные функции

В данном разделе описываются стандартные функции, которые могут использоваться в выражениях.

В описании стандартных функций понятие *числовой тип* означает один из типов INT (INTEGER) , SMALLINT, REAL или NUMERIC (DOUBLE).

Символьные функции

Символьные функции предназначены для обработки символьных выражений типа CHAR, VARCHAR..

Дополнение строки слева

Синтаксис

lpad (<строка>, <новая длина> [, <дополняемые символы>])

<строка> – выражение типа CHAR, VARCHAR.

<новая длина> – беззнаковый числовой литерал.

<дополняемые символы> – выражение типа CHAR, VARCHAR.

Описание

Функция дополняет строку заданными символами с левого края.

Если <новая длина> больше исходной длины <строки>, то <строка> расширяется слева <дополняемыми символами> до <новой длины> <строки> (возможно, с повторением <дополняемых символов>).

```
line:="12345";
new_line:=lpad(line, 12, "abc"); // abcabca12345
```

Если <дополняемые символы> не указаны, то по умолчанию <строка> дополняется пробелами.

Если значение <новая длина> меньше исходной длины <строки>, то исходная <строка> усекается до заданной <новой длины> справа.

```
line:="12345";
new_line:=lpad(line, 3, '**'); // 123
```

Если суммарная длина аргумента <дополняемые символы> и исходной длины <строки> больше, чем указанная <новая длина>, то <строка> дополняется только частью аргумента <дополняемые символы>. В этом случае аргумент <дополняемые символы> усекается справа.

```
line:="12345";
new_line:=lpad('12345',10,'abcdefgh'); // abcde12345
```

Возвращаемое значение

Возвращает <строку>, дополненную слева указанными последовательностями символов. Длина <строки> – максимум из длин <строки> и <новой длины>.

Тип возвращаемого значения совпадает с типом аргумента.

Если <строка> имеет NULL-значение, возвращается NULL-значение.

Дополнение строки справа

Синтаксис

rpad *(*<строка>, <новая длина> *[,* <дополняемые символы>*])*

<строка> – выражение типа CHAR, VARCHAR.

<новая длина> – беззнаковый числовой литерал.

<дополняемые символы> – выражение типа CHAR, VARCHAR.

Описание

Функция дополняет строку заданными символами с правого края.

Если <новая длина> больше исходной длины <строки>, то <строка> расширяется справа <дополняемыми символами> до <новой длины> <строки> (возможно, с повторением <дополняемых символов>).

Если <дополняемые символы> не указаны, то по умолчанию <строка> дополняется пробелами.

Если значение <новая длина> меньше исходной длины <строки>, то исходная <строка> усекается до заданной <новой длины> справа.

Если суммарная длина аргумента <дополняемые символы> и исходной длины <строки> больше, чем указанная <новая длина>, то <строка> дополняется только частью аргумента <дополняемые символы>. В этом случае аргумент <дополняемые символы> усекается справа.

Возвращаемое значение

Возвращает <строку>, дополненную справа указанными последовательностями символов.

Длина <строки> – максимум из длин <строки> и <новой длины>.

Тип возвращаемого значения совпадает с типом аргумента.

Если <строка> имеет NULL-значение, возвращается NULL-значение.

Пример

```
1)
   line:="Коньяк ";
  new_line:=rpad(line,12,"*"); // Коньяк *****

2)
line:="В горах сильное";
new_line:=rpad(line,len(line)+3*len(" эхо")," эхо");// В горах сильное эхо
   эхо эхо
```

Выделение конечной подстроки

Синтаксис

right_substr (<строка>,<количество>)

<строка> – выражение типа CHAR, VARCHAR.

<количество> – целое положительное число.

Описание

Выделяет правую часть <строки> заданного размера.

Возвращаемое значение

Длина возвращаемой строки – минимум из длин <строки> и <количеств*а*>.

Тип возвращаемого значения совпадает с типом аргумента.

Если <строка> имеет NULL-значение, возвращается NULL-значение.

Пример

```
1 Выделение функциональной части  из названия процедурных функций для
работы с BLOB-данными (Linter_Blob_Append, Linter_Blob__Get_Data и т. п.
Все функции имеют одинаковый префикс Linter_Blob)_

line:="Linter_Blob_Append";
new_line:=right_substr(line,len(line)-12); // Append


2 Преобразование  чисел  в формате аа… а.а…(n) в формат а.а…nnnnn
(например, 234.65(9) в 234.6599999,  .6(3)  в   633333)

line:="234.56(9)";
repeat:=right_substr(substr(line,1,len(line)-1),1);
new_line:=substr(line, 1, len(line)-3)+rpad("",5,repeat); // 234.5699999
```

Дублирование строки

Синтаксис

repeat_string (<строка>,<количество>)

<строка> – выражение типа CHAR, VARCHAR.

<количество> – целое положительное число.

Описание

Дублирование строки заданное число раз.

<Количество> должно быть константой (литералом).

Результирующая длина <строки> не должна превышать максимально допустимую длину для типа данных исходной <строки>.

Возвращаемое значение

Строка, являющаяся конкатенацией заданного <*количества*> исходной <*строки*>.

Пример

```
1)
   line:=repeat_string("*",10); // **********


2)
// Полем, полем, полем  ветер пролетел.
line:="Полем"+ repeat_string(", полем",2)+ " ветер пролетел.";
```

Поиск подстроки

Синтаксис

instr (<строка>,<подстрока> [,<начало поиска> [,<номер вхождения>]])

<строка> – выражение типа CHAR, VARCHAR.

<подстрока> – выражение типа CHAR, VARCHAR.

<начало поиска> – целое положительное число.

<номер вхождения>– целое положительное число.

Описание

Функция выполняет поиск подстроки в строке, начиная с заданной позиции и с учетом указанного номера вхождения.

Типы данных <строки> и <подстроки> должны быть приводимыми.

Длина <подстроки> не должна быть более 4000.

<Начало поиска> задает начальную позицию для поиска <подстроки>. Отсчет начинается с единицы. Если <начало поиска> не задано, по умолчанию принимается значение 1.

<Номер вхождения> задает порядковый номер искомой подстроки. Отсчет начинается с единицы. Если <номер вхождения> не задан, по умолчанию принимается значение 1.

Возвращаемое значение

Номер позиции, в которой размещается найденная в <строке> заданная <подстрока> или 0, если <строка> имеет нулевую длину, если подстрока не найдена или входные параметры имеют логически недопустимые значения.

Информация о недопустимых значениях входных параметров не возвращается.

Тип возвращаемого значения – INT.

Пример

```
declare
   var line char(50);
   var repeat int;
   var i int;
   code
   line:="Реляционная СУБД - это СУБД, которая ...";
   repeat:=2;
   i:=instr(line,"СУ"+"БД",1,repeat); // 24
```

Определение длины символьной строки

Синтаксис

Len | length|char_length (<строка>)

<строка> – выражение типа CHAR, VARCHAR.

Описание

Вычисляет длину символьной строки.

Возвращаемое значение

Возвращает числовое значение – длину <строки> в символах.

Если строка имеет NULL-значение, возвращается NULL-значение.

Тип возвращаемого значения – INT.

Пример

```
1

1)
line:="План-график";
i:=len(line); // 11


2)
i:=char_length(""); // 0


3)
line1:="abcd";
```

```
line2:="12345";
i:=char_length(line1+line2); // 9
```

Определение длины строки в байтах

Синтаксис

octet_length (<строка>)

<строка> – выражение типа CHAR, VARCHAR, NCHAR, NCHAR VARYING.

Описание

Определение длины строки в байтах.

Возвращаемое значение

Если <строка> имеет тип данных CHAR, VARCHAR, то возвращаемое значение аналогично функции len.

Если <строка> имеет тип данных NCHAR, NCHAR VARYING, то возвращаемое значение равно L*2, где L – длина <*строки*> в символах.

Тип возвращаемого значения – INT.

Пример

```
1)
i:=octet_length("\x34\x237\x06"); // 3

2)
line:="ASCII-строка";
i:=octet_length(line); // 12

3)
line:="UNICODE-строка";
i:=octet_length(tonchar(line)); // 28
```

Удаление крайних пробелов из символьной строки

Синтаксис

trim(<строка>)

<строка> – выражение типа CHAR, VARCHAR.

Описание

Удаляет из символьной строки пробелы справа и слева до первого отличного от пробела символа.

Пример

```
str:='  1.  Название колонки   \n  ';
str:=trim(str) //str='1.  Название колонки   \n'
```

Удаление символов из строки слева

Синтаксис

ltrim (<строка> [,<подстрока>])

<строка> – выражение типа CHAR, VARCHAR.

<подстрока> – выражение типа CHAR, VARCHAR.

Описание

Из <строки> удаляются слева символы, указанные в <подстроке>.

Если <подстрока> не задана, то по умолчанию удаляются пробелы.

Если <строка> имеет тип данных CHAR(1), VARCHAR(1) и удаляется содержащийся в ней символ, то <строка> не становится пустой: удаляемый символ заменяется пробелом, и длина строки остается равной 1 (таким образом, удаление пробела из поля типа CHAR(1), VARCHAR(1) с помощью функции LTRIM невозможно).

Возвращаемое значение

<Строка> с удаленными слева указанными символами.

Тип возвращаемого значения совпадает с типом аргумента.

Если аргумент NULL, результат NULL.

Пример

```
1)
   line:="***12345***";
   new_line:=ltrim(line,"*"); // 12345***

2)
  line:="*";
 new_line:=ltrim(line,"*");
  i:=length(new_line); // 0

3)
  line:=" ";
 new_line:=ltrim(line);
  i:=length(new_line); // 1
```

Удаление символов из строки справа

Синтаксис

rtrim (<строка> [,<подстрока>])

<строка> – выражение типа CHAR, VARCHAR.

<подстрока> – выражение типа CHAR, VARCHAR.

Описание

Из <строки> удаляются справа символы, указанные в <подстроке>.

Если <подстрока> не указана, то по умолчанию удаляются пробелы.

Если <строка> имеет тип данных CHAR(1), VARCHAR(1) и удаляется содержащийся в ней символ, то <строка> не становится пустой: удаляемый символ заменяется пробелом, и длина строки остается равной 1 (таким образом, удаление пробела из поля типа CHAR(1), VARCHAR(1) с помощью функции RTRIM невозможно).

Возвращаемое значение

<Строка> с удаленными справа указанными символами.

Тип возвращаемого значения совпадает с типом аргумента.

Если аргумент NULL, результат NULL.

Пример

```
1)
   line:="***12345***";
   new_line:=rtrim(line,"*"); // ***12345

2)
  line:="*";
 new_line:=rtrim(line,"*");
  i:=length(new_line); // 0

3)
  line:=" ";
 new_line:=rtrim(line);
  i:=length(new_line); // 1
```

Выделение символьной подстроки

Синтаксис

substr (<строка>,<нач поз>,<длина>)

<строка> – выражение типа CHAR, VARCHAR.

<нач поз> – целое положительное выражение не меньше 1.

<длина> – целое положительное выражение не меньше 0.

Описание

Возвращает подстроку из <строки>, которая начинается с символа с номером <нач поз> и имеет указанную <длина>. Если указана слишком большая длина, возвращаются все символы до конца исходной строки. Если <строка> имеет NULL-значение или является пустой, возвращается соответственно NULL-значение или пустая строка, независимо от остальных параметров. Если параметры <нач поз>, <длина> заданы неверно, возвращается пустая строка.

Примеры

```
str:="d.60-k.51";
str:=substr(str,3,2);         // 60

str:="format:3B-####.#";
str:=substr(str,8,len(str));  // 3B-####.#

str:=NULL;
str:=substr(str,5,200)        // NULL

str:= "";
str:=substr(str,5,200):       // ""

str:="d.60-k.51";
str:=substr(str,0,2);         // ""
str:=substr(str,-3,2);        // ""
```

```
str:=substr(str,2,-7);          // ""
```

Определение позиции подстроки

Синтаксис

strpos (<строка>,<подстрока> [,<справа>])

<строка> – выражение типа CHAR, VARCHAR.

<подстрока> – выражение типа CHAR, VARCHAR.

<справа> – выражение логического типа.

Описание

Функция ищет первое вхождение <подстроки> в заданной <строке> слева (или справа, если параметр <справа> задан и его значение – TRUE) и возвращает номер позиции исходной строки (начиная с 1), с которой начинается найденная подстрока. Если подстрока не найдена, возвращается 0. Поиск NULL-подстроки запрещен.

Пример

```
str:="пример поиска подстроки\n";
pos:=strpos(str,"поиск");              // 8
pos:=strpos(str,"\n",TRUE);            // 24
pos:=strpos(str,"");                   // 0
pos:=strpos(str,"по",FALSE);           // 8
pos:=strpos(str,"по",TRUE);            // 15
pos:=strpos(str,"примеры");            // 0
```

Корректировка подстроки

Синтаксис

insert (<строка>, <позиция> ,<длина>, <подстрока>)

<строка> – выражение типа CHAR, VARCHAR.

<позиция>– выражение целочисленного типа.

<длина>– выражение целочисленного типа.

<подстрока> – выражение типа CHAR, VARCHAR.

Описание

Корректировка подстроки в заданной строке (удаление подстроки или замена подстроки).

Типы данных <строки> и <подстроки> должны быть приводимыми.

Длина <подстроки> не должна быть более 4000.

<Позиция> задает позицию заменяемой подстроки в <строке>. Отсчет начинается с единицы.

<Длина> задает длину заменяемой подстроки в <строке>.

<Подстрока> задает значение, вставляемое вместо удаленной подстроки.

Начиная с <позиции>, удаляется <длина> символов, и вместо них вставляются символы <подстроки>.

Количество удаляемых символов не должно выходить за пределы строки, т.е. сумма значений <позиция> + <длина> должна быть не больше, чем значение «длина <строки> + 1».

Количество заменяемых символов может превышать количество удаляемых.

Возвращаемое значение

<Строка> с замененной <подстрокой>.

Код завершения при неправильных значениях аргументов функции.

Пример

```
1 формирование из строки 12345 строки 12**5
line:="12345";
new_line:=insert(line,3,2,"**"); // 12**5


2 формирование из строки 12345 строки 125
line:="12345";
new_line:=insert(line,3,2,""); // 125


3 формирование из строки 12345 строки 15ab
line:="12345";
new_line:=insert(insert(line,2,3,""),len(trim(insert(line,2,3,"")))+1,2,
"ab"); // 15ab


4 замена строки 12345 на строку abc

line:="12345";
new_line:=trim(insert(line,1,len(line),""))+"abc";
```

Замена всех подстрок

Синтаксис

replace (<строка>, <подстрока 1>, , <подстрока 2>)

<строка> – выражение типа CHAR, VARCHAR.

<подстрока 1> – выражение типа CHAR, VARCHAR.

<подстрока 2> – выражение типа CHAR, VARCHAR.

Описание

Замена всех подстрок в заданной строке.

Типы данных <строки>, <подстроки 1> и <подстроки 2> должны быть приводимыми.

Длина <подстроки 1>, <подстроки 2> не должна быть более 4000.

<Подстрока 1> задает удаляемое из <строки> значение.

<Подстрока 2> задает вставляемое вместо удаленной <подстрока 1> значение.

Возвращаемое значение

Исходная <строка>, в которой все вхождения <подстроки 1> заменены на <подстроку 2>.

Если значение <подстроки 1> в <строке> не найдено, <строка> возвращается без изменений.

Пример

```
line:="Имя таблицы PERSON";
line:=replace(line,toupper("person"),"\"Сотрудники\"");
// Имя таблицы "Сотрудники"
```

Замена символов строки

Синтаксис

translate (<строка>, <подстрока 1>, , <подстрока 2>)

<строка> – выражение типа CHAR, VARCHAR.

<подстрока 1> – выражение типа CHAR, VARCHAR.

<подстрока 2> – выражение типа CHAR, VARCHAR.

Описание

Замена указанных символов строки другими символами.

Типы данных <строки>, <подстроки 1> и <подстроки 2> должны быть приводимыми.

<Подстрока 1> задает набор заменяемых в <строке> символов.

<Подстрока 2> задает новые значения заменяемых символов.

Символы пробела, заданные в конце символьных выражений <строка>, <подстрока 1>, <подстрока 2> усекаются. Чтобы они принимались во внимание, необходимо использовать явное преобразование типа данных или не задавать пробелы в конце этих выражений.

Возвращаемое значение

Исходная <строка>, в которой каждый символ из <подстроки 1> заменен на соответствующий ему символ из <подстроки 2>. Например, если <подстрока 1>='ab', а <подстрока 2>='12', то каждый символ 'a' в исходной <строке> будет заменён на '1', а каждый символ 'b' в исходной <строке> – на '2';

Если <подстрока 1> длиннее <подстроки 2>, то все ее лишние символы удаляются из исходной <строки>, поскольку для них нет соответствующих символов в <подстроке 2>.

Если один из аргументов имеет значение NULL, результат будет NULL,

Примеры

```
1)
   line:="Важные события 20 века";

   line:=translate(line,"20","XX"); // Важные события XX века

2)

   line:="День недели 1 2 3 4 5 6 7";
   line:=translate(line,"1234567","пвсчпсв");
// День недели п в с ч п с в
```

Преобразование строки

Синтаксис

makestr (<строка>[, …])

<строка> – выражение типа CHAR, VARCHAR.

Описание

<Строка> может содержать знаки вопроса «?».

Возвращаемое значение

Возвращается строка, в которой вместо вопросов подставлены значения параметров, преобразованные к строке. Соответствие вопросов и параметров устанавливается по порядку.

Если необходимо включить в строку сам символ вопроса, он экранируется при помощи обратного слэша (\?).

Если количество вопросов и актуальных выражений-параметров не совпадает, возвращается NULL.

Пример

Функция удобна для формирования текста запроса, если использовать ее вместо конкатенации строк.

Например, вместо:

```
execute direct "insert into "+tabname+" values("+itoa(a*b)+",'"+
charValue+"','"+dtoa(dateValue)+"');";
```

можно писать:

```
execute direct makestr("insert into ? values(?, '?', '?');", tabname, a*b,
charValue, dateValue);
```

Такая конструкция читается лучше, не нужны вызовы функций преобразования типов (makestr их делает сама). В случае значения параметра NULL, makestr вставит текст «NULL».

Функция дает выигрыш в производительности по сравнению с функцией конкатенации строк.

Удвоение символа в строке

Синтаксис

dupchar (<строка>,<символ>)

Описание

Удвоение заданного <символа> в <строке>. Используется, как правило, для формирования текста запроса (удвоение апострофов).

Возвращаемое значение

Возвращается <строка>, в которой каждое вхождение <символа> удвоено.

Преобразование символов строки в коды

Синтаксис

chr (<строка>)

<строка> – значение типа BYTE.

Возвращаемое значение

Строка, каждый символ которой имеет код соответствующего элемента <строки>. Длина возвращаемой строки равна длине <строка> или меньше, если в <строке> встречается нулевой байт.

Преобразование из символьного вида в шестнадцатеричный

Синтаксис

asc (<строка1>,<строка2>)

<строка1> – значение типа CHAR или NCHAR;

<строка2> – значение типа BYTE.

Возвращаемое значение

Функция формирует в <строке2> типа BYTE шестнадцатеричные коды символов из <строки1>. Количеством формируемых байтов определяется длина <строки2>. Если длина <строки1> меньше длины <строки2>, остаток <строки2> заполняется нулями.

Если <строка1> имеет тип данных NCHAR, то в <строку2> заносятся 2-х байтовые коды символов <строки1>.

Пример

Добавление в таблицу UNICODE-значения `unic_var` независимо от текущих кодировок:

```
asc(unic_var, out);
execute direct "insert into t(uc)
values (hex('"+btoa(out)+"'));";
```

Преобразование строки к верхнему регистру

Синтаксис

toupper|upper (<строка>)

<строка> – значение типа CHAR, VARCHAR .

Возвращаемое значение

<Строка>, в которой все символы имеют заглавное (прописное) представление, т.е. буквы алфавита a-z, а-я преобразованы в A-Z, А-Я.

Тип возвращаемого значения совпадает с типом аргумента.

Если аргумент NULL, то результат NULL.

Преобразование строки к нижнему регистру

Синтаксис

tolower|lower (<строка>)

<строка> – значение типа CHAR, VARCHAR.

Возвращаемое значение

<Строка>, в которой все символы имеют строчное представление, т.е. буквы алфавита A-Z, А-Я преобразованы в a-z ,а-я.

Тип возвращаемого значения совпадает с типом аргумента.

Если аргумент NULL, результат NULL.

Перевод начальной буквы слова в заглавную

Синтаксис

initcap (<строка>)

<строка> – выражение типа CHAR, VARCHAR.

Описание

Перевод первой буквы каждого слова строки в заглавную.

Разделителями слов в <строке> являются все коды со значением не больше кода пробела.

Возвращаемое значение

Строка того же типа и длины.

Если аргумент является NULL значением, возвращается NULL.

Пример

```
1)
   line:="Организация объединённых наций";
   line:=initcap(line); // Организация Объединённых Наций

2)
  line:="a\nb\n\c\nd";
  new_line:=initcap(line); //  A\nB\n\C\nD
```

Корректировка подстроки

Синтаксис

insert (<строка>, <позиция> ,<длина>, <подстрока>)

<строка> – выражение типа CHAR, VARCHAR.

<позиция>– выражение целочисленного типа.

<длина>– выражение целочисленного типа.

<подстрока> – выражение типа CHAR, VARCHAR.

Описание

Корректировка подстроки в заданной строке (удаление подстроки или ее замена).

Типы данных <строки> и <подстроки> должны быть приводимыми.

Длина <подстроки> не должна быть более 4000.

<Позиция> задает позицию заменяемой подстроки в <строке>. Отсчет начинается с единицы.

<Длина> задает длину заменяемой подстроки в <строке>.

<Подстрока> задает значение, вставляемое вместо удаленной подстроки.

Начиная с <позиции>, удаляется <длина> символов, и вместо них вставляются символы <подстроки>.

Количество удаляемых символов не должно выходить за пределы строки, т.е. сумма значений <позиция> + <длина> должна быть не больше, чем значение «длина <строки> + 1».

Количество заменяемых символов может превышать количество удаляемых.

Возвращаемое значение

<Строка> с замененной <подстрокой>.

Код завершения при неправильных значениях аргументов функции.

Пример

```
1)
   line:="Дата рождения: ";
   new_line:=insert(line,  len(line),10,"04.05.2006");  //  Дата  рождения:
04.05.2006
2)
   line:="Coca-Cola";
   new_line:=insert(line,1,4,"Pepsi"); // Pepsi-Cola
```

Преобразование числового выражения в символьный вид

Синтаксис

to_char (<значение>[,<формат>])

<значение> – значение типа SMALLINT, INTEGER, BIGINT, NUMERIC, REAL или DOUBLE.

<формат> – строковый литерал, задающий формат символьного представления в виде шаблона:

[FM][<S>][<$>][{<знак>{I,}[<.><знак>{I,}]} | {<цифра>{I,}[.<цифра>{I,}]EEEE>}]

<знак>::= <цифра> | <,>

<цифра>::= '0'|'9'

'0' – принудительный вывод незначащих нулей

'9' – принудительный вывод ведущих пробелов вместо незначащих нулей и незначащих нулей после запятой.

FM – вывод без ведущих пробелов

S – принудительный вывод знака '+'/'-'

$ – принудительный вывод '$'

. – децимальная точка

, – разделитель групп цифр в целой части выводимого значения.

X – вывод в шестнадцатеричной форме.

EEEE – вывод в экспоненциальной форме.

Ограничения:

- до запятой – не более 32 позиций;
- после запятой – не более 15 позиций;
- групп цифр – не более 32.

Примеры

```
  c:=to_char(123I, "XX");
  execute direct "insert into test_to_char_res values(' 7B','"+c+"');";
  c:=to_char(2063597568I, "XXXXXXXXX");
  execute direct "insert into test_to_char_res values('
7B000000','"+c+"');";
  c:=to_char(123B, "XX");
  execute direct "insert into test_to_char_res values(' 7B','"+c+"');";
  c:=to_char(2063597568B, "XXXXXXXXX");
  execute direct "insert into test_to_char_res values('
7B000000','"+c+"');";
  c:=to_char(34621422143503227, "XXXXXXXXXXXXXXX");
  execute direct "insert into test_to_char_res values('
7B0000007B7B7B','"+c+"');";

  c:=to_char(+123.45N, "XX");
  execute direct "insert into test_to_char_res values(' 7B','"+c+"');";
  c:=to_char(+2063597568.45N, "XXXXXXXXX");
  execute direct "insert into test_to_char_res values('
7B000000','"+c+"');";
  c:=to_char(-123.45N, "XXX");
  execute direct "insert into test_to_char_res values('####','"+c+"');";


//  /* EEEE */
  c:=to_char(+123.456N, "9.9EEEE");
  execute direct "insert into test_to_char_res values('
1.2E+02','"+c+"');";
  c:=to_char(+123.456N, "FM9.9EEEE");
  execute direct "insert into test_to_char_res values('1.2E+02','"+c+"');";

  c:=to_char(1000.0D, "FM9.9EEEE"); //+1E+03D
  execute direct "insert into test_to_char_res values('1.E+03','"+c+"');";
  c:=to_char(-0.001D, "FM9.9EEEE"); //-1E-03D
  execute direct "insert into test_to_char_res values('-1.E-03','"+c+"');";

  c:=to_char(1.1111D, "S$9EEEE");
  execute direct "insert into test_to_char_res values(' +$1E+00','"+c+"');";
  c:=to_char(-1.1111D, "S$9EEEE");
  execute direct "insert into test_to_char_res values(' -$1E+00','"+c+"');";
```

```
  c:=to_char(-1.1111D, "FMS$9EEEE");
  execute direct "insert into test_to_char_res values('-$1E+00','"+c+"');";
  c:=to_char(-1.1121D, "FMS$9.999EEEE");
  execute direct "insert into test_to_char_res values('-
$1.112E+00','"+c+"');";
```

Фонетический код строки

Синтаксис

SOUNDEX (<значение>)

<значение> - выражение типа CHAR, VARCHAR

Описание

Получить фонетический код значения.

Возвращаемое значение

Символьная строка, представляющая фонетический код заданного <значения>.

Тип результата – CHAR(4).

Возвращаемое значение в дальнейшем может использоваться для подбора подходящей фонетической фразы.

Примеры

1)

Фонетические коды 0124 указывают на близкое фонетическое звучание слов «опечатка» и «опичатка» .

```
new_line:=soundex("опечатка"); // O124

new_line:=soundex("опичатка"); // O124
```

2)

Фонетические коды B700 указывают на близкое фонетическое звучание фраз в предложении «Вы ли выли ?» .

```
new_line:=soundex("выли"); // B700
new_line:=soundex("вы ли"); // B700
```

3)

```
new_line:=soundex("пень"); // П900
new_line:=soundex("лень"); // Л900
```

Определение близости фонетического звучания строк

Синтаксис

difference (<значение 1>, <значение 2>)

<значение 1> - выражение типа CHAR, VARCHAR

<значение 2> - выражение типа CHAR, VARCHAR

Описание

Определение близости фонетического звучания.

Возвращаемое значение

Разность фонетического звучания двух аргументов, вычисляемая на основании кодов фонетического значения аргументов (см. функцию SOUNDEX);

Тип результата – INTEGER.

Значение 0 указывает на фонетическое совпадение аргументов, 1 – на существенное различие.

Пример

```
i:=difference("столб", "столп"); // 0

i:=difference("ошибка", "ашипка"); // 1
```

Функции для работы с UNICODE-значениями

Данные функции предназначены для обработки UNICODE-значений. Все они, кроме функции `nlen`, возвращают результат типа NCHAR.

Определение длины UNICODE-строки

Синтаксис

nlen (<строка>)

<строка> – выражение типа NCHAR, NVARCHAR.

Описание

Вычисляет длину UNICODE-строки.

Возвращаемое значение

Возвращает числовое значение – длину <строки> в UNICODE-символах. Если строка имеет NULL-значение, возвращается NULL-значение.

Удаление пробелов UNICODE-строки

Синтаксис

ntrim (<строка>)

<строка> – выражение типа NCHAR, NVARCHAR.

Описание

Удаляет из строки пробелы справа и слева до первого отличного от пробела символа.

Выделение UNICODE–подстроки

Синтаксис

nsubstr (<строка>,<нач поз>,<длина>)

<строка> – выражение типа NCHAR, NVARCHAR.

<нач поз> – целое положительное выражение не меньше 1.

<длина> – целое положительное выражение не меньше 0.

Описание

Возвращает подстроку из <строки>, которая начинается с символа с номером <нач поз> и имеет указанную <длину>. Если указана слишком большая длина, возвращаются все символы до конца исходной строки. Если <строка> имеет NULL-значение или является пустой, возвращается, соответственно, NULL-значение или пустая строка, независимо от остальных параметров. Если параметры <нач поз>, <длина> заданы неверно, возвращается пустая строка.

Дополнение UNICODE–строки слева

Синтаксис

nlpad (<строка>, <новая длина> [, <дополняемые символы>])

<строка> – выражение типа NCHAR, NVARCHAR.

<новая длина> – беззнаковый числовой литерал.

<дополняемые символы> – выражение типа NCHAR, NVARCHAR.

Описание

Функция дополняет строку заданными символами с левого края.

См. описание функции lpad.

Дополнение UNICODE–строки справа

Синтаксис

nrpad (<строка>, <новая длина> [, <дополняемые символы>])

<строка> – выражение типа NCHAR, NVARCHAR.

<новая длина> – беззнаковый числовой литерал.

<дополняемые символы> – выражение типа NCHAR, NVARCHAR.

Описание

Функция дополняет строку заданными символами с правого края.

См. описание функции rpad.

Выделение последних символов UNICODE–строки

Синтаксис

nright_substr (<строка>,<количество>)

<строка> – выражение типа NCHAR, NVARCHAR.

<количество> – целое положительное число.

Описание

Выделяет правую часть <строки> заданного размера.

См. описание функции right_substr.

Дублирование UNICODE–строки

Синтаксис

nrepeat_string (<строка>,<количество>)

<строка> – выражение типа NCHAR, NVARCHAR.

<количество> – целое положительное число.

Описание

Дублирование строки заданное число раз.

См. описание функции repeat_string.

Выделение позиции подстроки

Синтаксис

nstrpos (<строка>,<подстрока> [,<справа>])

<строка> – выражение типа NCHAR.

<подстрока> – выражение типа NCHAR.

<справа> – выражение логического типа.

Описание

Функция ищет первое вхождение <подстроки> в заданной <строке> слева (или справа, если параметр <справа> задан и его значение – TRUE) и возвращает номер позиции исходной строки (начиная с 1), с которой начинается найденная подстрока. Если подстрока не найдена, возвращается 0. Поиск NULL-подстроки запрещен.

Преобразование строки к верхнему регистру

Синтаксис

ntoupper | nupper (<строка>)

<строка> – значение типа NCHAR, NVARCHAR.

Возвращаемое значение

Преобразованное к верхнему регистру значение <строки>.

Преобразование строки к нижнему регистру

Синтаксис

Ntolower| nlower (<строка>)

<строка> – значение типа NCHAR, NVARCHAR.

Возвращаемое значение

Преобразованное к нижнему регистру значение <строки>.

Преобразование в UNICODE-строку

Синтаксис

tonchar (<значение> [, <параметры>])

<значение> – выражение любого допустимого типа.

Описание

Выполняет универсальное преобразование любого типа в UNICODE-строку. Значение <параметры> зависит от типа <значения> и соответствует дополнительным параметрам в функциях itoa, ftoa, ntoa, dtoa, btoa (например, формат для dtoa, точность для ntoa и т.д.).

Для параметра типа char выполняется перекодировка значения символьного типа (кодировка значения символьного типа определяется прозрачно для пользователя, как это описано на стр. 46, «Поддержка кодовых страниц»).

Удаление символов из UNICODE–строки слева

Синтаксис

nltrim (<строка> [,<подстрока>])

<строка> – выражение типа NCHAR, NVARCHAR.

<подсlтрока> – выражение типа NCHAR, NVARCHAR.

Описание

Из <строки> удаляются слева символы, указанные в <подстроке>.

См. описание функции ltrim.

Удаление символов из UNICODE–строки справа

Синтаксис

nrtrim (<строка> [,<подстрока>])

<строка> – выражение типа NCHAR, NVARCHAR.

<подстрока> – выражение типа NCHAR, NVARCHAR.

Описание

Из <строки> удаляются справа символы, указанные в <подстроке>.

См. описание функции rtrim.

Перевод начальной буквы UNICODE–слова в заглавную

Синтаксис

ninitcap (<строка>)

<строка> – выражение типа NCHAR, NVARCHAR.

Описание

Перевод первой буквы каждого слова строки в заглавную.

См. описание функции initcap.

Корректировка UNICODE–подстроки

Синтаксис

ninsert (<строка>, <позиция> ,<длина>, <подстрока>)

<строка> – выражение типа NCHAR, NVARCHAR.

<позиция>– выражение целочисленного типа.

<длина>– выражение целочисленного типа.

<подстрока> – выражение типа NCHAR, NVARCHAR.

Описание

Корректировка подстроки в заданной строке (удаление подстроки или ее замена).

См. описание функции insert.

Замена всех UNICODE–подстрок

Синтаксис

nreplace (<строка>, <подстрока 1>, , <подстрока 2>)

<строка> – выражение типа NCHAR, NVARCHAR.

<подстрока 1> – выражение типа NCHAR, NVARCHAR.

<подстрока 2> – выражение типа NCHAR, NVARCHAR.

Описание

Замена всех подстрок в заданной строке.

См. описание функции replace.

Замена символов UNICODE–строки

Синтаксис

```
ntranslate (<строка>, <подстрока 1>, , <подстрока 2>)
```

<строка> – выражение типа NCHAR, NVARCHAR.

<подстрока 1> – выражение типа NCHAR, NVARCHAR.

<подстрока 2> – выражение типа NCHAR, NVARCHAR.

Описание

Замена указанных символов строки другими символами.

См. описание функции translate.

Математические функции

Вычисление абсолютного значения

Синтаксис

```
abs (<значимое числовое выражение>)
```

Описание

Вычисление абсолютного значения числа.

Возвращаемое значение

Тип возвращаемого значения совпадает с типом аргумента.

Если аргумент NULL, результат NULL.

Округление до целого с избытком

Синтаксис

```
ceil (<значимое числовое выражение>)
```

Описание

Округление числа до ближайшего верхнего целого значения.

Возвращаемое значение

Целое число – результат округления (до целого) с избытком <значимого числового выражения>. Тип возвращаемого значения – DOUBLE. Функция возвращает наименьшее целое значение, большее или равное аргументу.

Если аргумент NULL, результат NULL.

Округление до целого с недостатком

Синтаксис

floor (<значимое числовое выражение>).

Описание

Округление числа до ближайшего нижнего целого значения.

Возвращаемое значение

Целое число – результат округления (до целого) с недостатком <значимого числового выражения>. Тип возвращаемого значения – DOUBLE.

Если аргумент NULL, результат NULL.

Округление значения типа «дата-время»

Синтаксис

date_round (<выражение>,<формат>).

<выражение> – выражение типа «дата-время».

<формат> – элемент формата «дата-время»:

{‘**D**’ | ‘**M**’ | ‘**Y**’ |‘**HH**' | '**HH12**'| '**HH24**'| '**MI**'| '**SS**'}

Описание

Округление даты до заданного значения.

Округление по году и месяцу имеет свою специфику, например:

```
dt:=date_round(sysdate(),"y"); //01.01.2007:00:00:00.00
```

т. е. выдается день и месяц.

«округленный"» год предлагается выбирать так:

```
var i int;
i:=year(sysdate()); // 2006
```

Возвращаемое значение

<Выражение>, округленное до заданной точности.

Тип возвращаемого результата – DATE.

Если аргумент NULL, результат NULL.

Пример

```
dt:=date_round(atod("28.04.2000","dd.mm.yyyy"),"m");//01.05.2000:00:00:00.00
```

Усечение представления значения типа «дата-время»

Синтаксис

date_trunc (<выражение>,<точность>).

<выражение> – выражение типа «дата-время».

<точность> – элемент формата «дата-время»:

{‘D’ | ‘M’ | ‘Y’ |‘HH' | 'HH12'| 'HH24'| 'MI'| 'SS'}

Описание

Усечение значения с заданной точностью.

Элемент формата «дата-время» задает точность усечения значения типа «дата-время» (до какого элемента даты-времени должно выполняться усечение).

Возвращаемое значение

<Выражение>, усеченное до заданной точности;

Тип возвращаемого результата – DATE;

Если аргумент <*выражение*> NULL, результат NULL.

Пример

```
// Sysdate =27.09.2006:15:45:34.22
   dt:=date_trunc(sysdate(), "m"); // 01.09.2006:00:00:00.00
```

Тригонометрические функции

Синтаксис

cos(<значимое числовое выражение>)
sin(<значимое числовое выражение>)
tan(<значимое числовое выражение>)

<значимое числовое выражение> – угол, выраженный в радианах

Описание

Вычисление тригонометрических функций.

Возвращаемое значение

Значение косинуса (COS), синуса (SIN) и тангенса (TAN) <значимого числового выражения> в радианах. Тип возвращаемого результата – DOUBLE.

Если аргумент NULL, результат NULL.

Обратные тригонометрические функции

Синтаксис

acos(<значимое числовое выражение>)
asin(<значимое числовое выражение>)
atan(<значимое числовое выражение>)
atan2(<значимое числовое выражение 1>, <значимое числовое выражение 2>)

<значимое числовое выражение> – угол, выраженный в радианах

Описание

Вычисление обратных тригонометрических функций.

Тип данных <значимого числового выражения> должен быть DOUBLE или приводиться к нему.

Возвращаемое значение

Значение обратного косинуса (ACOS), обратного синуса (ASIN), обратного тангенса (ATAN) от <значимое числовое выражение> в радианах и значение обратного тангенса частного <значимое числовое выражение 1>./<значимое числовое выражение (ATAN2).

Тип возвращаемого результата – DOUBLE.

Если аргумент NULL, результат NULL.

Гиперболические функции

Синтаксис

cosh(<значимое числовое выражение>)
sinh(<значимое числовое выражение>)
tanh(<значимое числовое выражение>)

<значимое числовое выражение> – угол, выраженный в радианах

Описание

Вычисление гиперболических функций.

Возвращаемое значение

Значение гиперболического косинуса (COSH), синуса (SINH) и тангенса (TANH) <значимого числового выражения> в радианах. Тип возвращаемого результата – DOUBLE.

Если аргумент NULL, результат NULL.

Экспоненциальная функция

Синтаксис

```
exp(<значимое числовое выражение>)
```

Описание

Вычисление экспоненциального значения.

Возвращаемое значение

Экспоненциальное значение <значимого числового выражения>. Тип возвращаемого результата – DOUBLE.

Если аргумент NULL, результат NULL.

Логарифмические функции

Синтаксис

```
ln  (<значимое числовое выражение>)
log (<значимое числовое выражение>,<основание>)
```

<основание> ::= вещественный литерал

Описание

Вычисление гиперболических функций.

Возвращаемое значение

Функция LN возвращает натуральный логарифм <значимого числового выражения> .

Функция LOG возвращает логарифм <значимого числового выражения> по заданному <основанию>.

Если значение <значимого числового выражения> или <основания> меньше или равно 0, или значение основания равно 1, возбуждается исключение BADPARAM.

Тип возвращаемого результата в обоих случаях – DOUBLE.

Если аргумент NULL, результат NULL.

Округление с заданной точностью

Синтаксис

```
round(<значимое числовое выражение>,<точность>)

<точность> ::= <значимое числовое выражение>
```

<Точность> задает точность округления:

- при положительном значении – точность округления после десятичной точки (количество цифр после десятичной точки);
- при отрицательном значении – точность округления перед десятичной точкой, т.е. при «–1» до ближайшего целого десятка («–2» – до сотни, «–3» до тысячи и т.д.).

Описание

Округление числа с заданной точностью.

Возвращаемое значение

<Значимое числовое выражение>, округленное до заданной точности.

Если <значимое числовое выражение> не может быть округлено с заданной точностью перед десятичной точкой, то возвращается ноль (например, задано округление 45.67 до тысяч – round(45.67,–3)).

Тип возвращаемого результата – DOUBLE.

Если аргумент NULL, результат NULL.

Усечение числа с заданной точностью

Синтаксис

trunc(<значимое числовое выражение>,<точность>)

<точность> ::= <значимое числовое выражение>

Описание

Усечение числа с заданной точностью.

При усечении <значимого числового выражения> округление не выполняется.

<Точность> задает точность усечения: при положительном значении усечение выполняется после десятичной точки (количество цифр после десятичной точки). Отрицательное значение – точность округления перед десятичной точкой (т.е. при –1 до ближайшего целого десятка, –2 – до сотни, –3 до тысячи и т.д.).

Возвращаемое значение

<Значимое числовое выражение>, округленное до заданной точности. Тип возвращаемого результата – DOUBLE.

Если аргумент NULL, результат NULL.

Определение знака числа

Синтаксис

sign(<значимое числовое выражение>)

Описание

Определение знака числа.

Возвращаемое значение

- –1 – <значимое числовое выражение> отрицательное;
- 1 – <значимое числовое выражение> положительное;
- 0 – <значимое числовое выражение> равно нулю.

Тип возвращаемого результата – INT.

Вычисление квадратного корня числа

Синтаксис

sqrt(<значимое числовое выражение>)

Описание

Вычисление квадратного корня числа.

Возвращаемое значение

Корень квадратный из <значимого числового выражения>. Тип возвращаемого результата – DOUBLE.

Если аргумент отрицательный, вызывается исключение BADPARAM.

Если аргумент NULL, результат NULL.

Работы с датами

Выделение дня из даты

Синтаксис

day (<дата>)

где <дата> – выражение типа DATE.

Описание

Возвращает номер дня из <даты>.

Пример

```
cur_dat:=17.11.1997:18:25:47.88;
num_day:=day(date);                // 17
```

Выделение месяца из даты

Синтаксис

month (<дата>)

где <дата> – выражение типа DATE.

Описание

Возвращает номер месяца из <даты>.

Пример

```
cur_dat:=17.11.1997:18:25:47.88;
num_month:=month(date);            // 11
```

Выделение года из даты

Синтаксис

year (<дата>)

где <дата> – выражение типа DATE.

Описание

Возвращает номер года из <даты>.

Пример

```
cur_dat:=17.11.1997:18:25:47.88;
num_year:=year(date);              // 1997
```

Выделение часа из даты

Синтаксис

hour (<дата>)

где <дата> – выражение типа DATE.

Описание

Возвращает номер часа из <даты>.

Пример

```
cur_dat:=17.11.1997:18:25:47.88;
num_hour:=hour(date);              // 18
```

Выделение минут из даты

Синтаксис

minute (<дата>)

где <дата> – выражение типа DATE.

Описание

Возвращает номер минуты из <даты>.

Пример

```
cur_dat:=17.11.1997:18:25:47.88;
num_minute:=minute(date);          // 25
```

Выделение секунд из даты

Синтаксис

second (<дата>)

где <дата> – выражение типа DATE.

Описание

Возвращает номер секунды из <даты>.

Пример

```
cur_dat:=17.11.1997:18:25:47.88;
num_second:=second(date); // 47
```

Выделение тиков из даты

Синтаксис

ticks (<дата>)

где <дата> – выражение типа DATE.

Описание

Возвращает номер тика из <даты>.

Пример

```
cur_dat:=17.11.1997:18:25:47.88;
num_ticks:=ticks(date);              // 88
```

Формирование даты

Синтаксис

make_date (<день>,<месяц>,<год>,<час>,<мин>,<сек>,<тики>)

где <день>, <месяц>, <год>, <час>, <мин>, <сек>, <тики> - выражения числового типа.

Описание

Функция возвращает значение типа DATE, сформированное для указанных дня, месяца, года, часа, минут и секунд с тиками. Любое количество параметров времени справа может быть не задано, в этом случае вместо них предполагаются нули. Если значения параметров недопустимы, функция возвращает нулевую дату, то есть 0.0.0:0:0:0.0.

Получение текущей даты

Синтаксис

sysdate ()

Описание

Получение текущей даты.

Возвращаемое значение

Текущая системная дата, с учетом установленной временной зоны.

Последний день месяца

Синтаксис

last_day (<значимое выражение>)

<значимое выражение> – выражение типа DATE.

Описание

Вычисление последнего дня месяца для указанной даты.

Возвращаемое значение

Возвращается значение типа DATE, представляющее дату последнего дня того месяца, который выбран из аргумента функции..

Пример

```
d:=last_day(sysdate()); // 30.09.2006:10:24:12
```

Дата очередного дня недели

Синтаксис

next_day (<значимое выражение>,<день недели>)

<значимое выражение> – выражение типа DATE.

<день недели> – символьное выражение или приводимое к нему, которое должно иметь одно из следующих значений:

Название дня недели		
полное	**сокращенное**	
Monday	Mon	Понедельник
Tuesday	Tue	Вторник
Wednesday	Wed	Среда
Thursday	Thu	Четверг
Friday	Fri	Пятница
Saturday	Sat	Суббота
Sunday	Sun	Воскресенье

Описание

Вычисление даты очередного дня недели.

Значение времени в возвращаемой дате совпадает с аналогичным значением в исходной дате.

Если запрашиваемый день недели совпадает с днем недели в исходной дате, то возвращается дата следующего (т. е. через 7 дней) дня недели.

Возвращаемое значение

Возвращается значение типа DATE, соответствующее указанному <дню недели> после заданной даты.

Пример

```
// sysdate=25.09.2006
d:=next_day(sysdate(),"mon"); // 02.10.2006
```

Помесячное изменение даты

Синтаксис

add_months (<значимое выражение>,<количество месяцев>)

<значимое выражение> – выражение типа DATE.

<количество месяцев> – численное значение типа INT, SMALLINT, BIGINT NUMERIC, REAL, DOUBLE или приводимое к нему.

Описание

Арифметическое добавление месяцев к исходной дате.

<Значимое выражение> должно иметь тип DATE или приводиться к нему.

При положительном значении аргумента <количество месяцев> формируется будущая дата, при отрицательном – прошлая по сравнению с исходной.

Если значение параметра <количество месяцев> не является целочисленным значением, то оно усекается до целой части.

При добавлении месяцев номер дня в результирующей дате не меняется, за исключением тех случаев, когда он приходится на конец месяца.

Возвращаемое значение

Возвращается значение типа DATE, увеличенное (уменьшенное) на заданное <количество месяцев>.

Пример

```
// sysdate=25.09.2006
d:=add_months(sysdate(),5); // 25.02.2007
```

Выделение заданных элементов даты

Синтаксис

datesplit (<значимое выражение>,<параметр>)

<значимое выражение> – выражение типа DATE.

<параметр> – <односимвольный литерал> | <двухсимвольный литерал>

Описание

<Значимое выражение> должно быть представлено в одном из форматов значений типа DATE или в виде литерала типа <дата-время> в формате по умолчанию .

<Параметр> определяет возвращаемое функцией значение.

Допустимы следующие значения <параметра>:

Значение параметра	Возвращаемое значение
“D”	День месяца
“M”	Номер месяца

Значение параметра	Возвращаемое значение
“QY”	Номер квартала
“Y”	Год
“DW”	Номер дня недели
“DY”	Номер дня в году
“WM”	Номер недели в месяце
“WY”	Номер недели в году
“ND”	Номер дня от начала нашей эры
“NW”	Номер недели от начала нашей эры
“NM”	Номер месяца от начала нашей эры
“HH”	Количество часов (диапазон 00-23)
“HH12”	Количество часов (диапазон 0-12)
“HH24”	Количество часов (диапазон 00-23)
“MI”	Количество минут
“SS”	Количество секунд
“FF”	Количество тиков

Возвращаемое значение

Возвращается указанный элемент <значимого выражения>.

Тип возвращаемого значения – INT.

Пример

```
// sysdate=10.04.2006
d:=datesplit(sysdate(),”m”); // 4
d:=datesplit(sysdate(),”qy”); // 2
```

Изменение даты на заданный интервал времени

Синтаксис

multime (<тип интервала>, <интервал>, <исходная дата>)

<исходная дата> – значение типа DATE.

<интервал> – целочисленное значение.

<тип интервала> – положительное целочисленное значение, задающее единицу измерения интервала времени.

Допустимые значения параметра <тип интервала>:

Значение	Интервал времени
1	Тики

Значение	Интервал времени
2	Секунды
4	Минуты
8	Часы
16	Дни
32	Недели
64	Месяцы
128	Кварталы
256	Годы

Значение <интервала> не должно задавать дату более 2099 года.

Возвращаемое значение

Возвращается значение типа DATE, увеличенное (уменьшенное) по сравнению с <исходной датой> на заданный <интервал>.

Если <исходная дата> представлена только временем, и <тип интервала> задает дни, недели, месяцы, кварталы или годы, то она перед вычислением устанавливается к текущей дате.

Пример

```
// sysdate=10.04.2008
dt:=multime(64,1,sysdate()); //10.05.2008

dt:=17.11.1997:18:25:47.88; //
dt:=multime(8,-15, dt); // 17.11.1997:03:25:47.88

// sysdate=10.04.2008
dt:=multime(16,3, atod("10","hh")); // 13.04.2008:10:00:00.00
```

Вычисление интервала между двумя датами

Синтаксис

divtime (<тип интервала>, <начальная дата>, <конечная дата>)

<начальная дата> – значение типа DATE.

<конечная дата> – значение типа DATE.

<тип интервала> – см. описание функции MUITIME.

Возвращаемое значение

Возвращается значение типа INT, представляющее разницу между конечной и начальной датами в единицах измерения, заданных параметром <тип интервала>.

Округление происходит в меньшую сторону. Например, если <тип интервала> = 256 (годы), а <начальная дата> больше <конечной даты> хотя бы на один тик, будет возвращено значение – 1.

Пример

```
dt_begin:=17.11.1997:18:25:47.88; //
dt_end:=17.11.1997:20:25:47.88;; //
i:=divtime(8,dt_begin, dt_end); //  2
```

Функции преобразования типов

Функции преобразования типов получают один параметр некоторого типа и возвращают значение, преобразованное к другому типу.

Представление числа в символьном виде

Синтаксис

itoa (<число>)

<число> – выражение числового типа.

Описание

Возвращается символьное представление <числа> со знаком (при этом знак «+» опускается). Если <число> не является целым, то его дробная часть игнорируется.

Примеры

```
str_num:=itoa(0);            // '0'
str_num:=itoa(NULL);         // 'NULL'
str_num:=itoa(123);          // '123'
str_num:=itoa(-27);          // '-27'
str_num:=itoa(3.1415);       // '3'
str_num:=itoa(77.);          // '77'
```

Представление числа в символьном виде с учетом знака

Синтаксис

ftoa (<число>)

<число> – выражение числового типа.

Описание

Возвращается символьное представление <числа> со знаком. Параметр <число> трактуется как вещественное число. В процессе преобразования применяются следующие правила:

- для положительных чисел знак «+» не формируется;
- общее число цифр в сформированной строке, включая целую, дробную части и разделительную точку – не больше 7;

- если число не умещается в диапазон преобразования, то оно округляется, незначащие нули отбрасываются. Если в диапазон не помещается целая часть числа, то результат функции представляется в виде <мантисса><порядок> (то есть <мантисса>, умноженная на 10 в степени <порядок>), где <мантисса> занимает 7 знаков, <порядок> – 3 знака.

Примеры

```
str_num:=ftoa(5);               // '5'

str_num:=ftoa(NULL);            // 'NULL'

str_num:=ftoa(09999.348);       // '9999.35'

str_num:=ftoa(0.0001);          // '0.0001'

str_num:=ftoa(-27.12387);       // '-27.1239'

str_num:=ftoa(3.1415);          // '3.1415'

str_num:=ftoa(77.);             // '77'

str_num:=ftoa(089.56);          // '89.56'

str_num:=ftoa(89.5600);         // '89.56'
```

Представление числа в символьном виде с учетом знака и заданной точности

Синтаксис

ntoa (<число>,[<длина>,[<точность>]]),

<число>, <длина> и <точность> – выражения числового типа.

Описание

Возвращается символьное представление <числа> со знаком и с заданной точностью. Параметр <число> трактуется как вещественное число. Параметр <длина> задает общую длину символьной строки с учетом знакового разряда числа и разделительной точки, параметр <точность> – количество цифр после запятой. По умолчанию длина предполагается равной 30, а точность – 10. Если точность равна 0, дробная часть и десятичная точка не выводятся. Для этой функции действуют те же правила преобразования (кроме диапазона представления), что и для функции FTOA. Правила преобразования для диапазона: если <длина> не достаточна для того, чтобы отобразить знак, целую часть числа, десятичную точку и указанное количество знаков после запятой, результатом является строка из символов «*» (их количество равно параметру <длина>). Если длины достаточно, отображается целая часть, десятичная точка и нужное количество знаков после запятой, причем нули после запятой не отбрасываются. Строка всегда имеет длину, равную параметру <длина>. Если число значащих символов меньше этого параметра, строка дополняется слева пробелами.

Примеры

```
str_num:=ntoa(1.23,10,3);        // '     1.230'

str_num:=ntoa(123.45, 5, 0);     // '  123'
```

Представление даты в символьном виде

Синтаксис

dtoa (<дата> [,<формат представления>]),

<дата> – выражение типа DATE.

<формат представления> – логическое выражение или строковый литерал.

Строковый литерал должен задавать формат преобразования <даты> в символьный вид. Для спецификации форматной строки можно использовать следующие обозначения:

- DD, dd – день;
- MM, mm, Mon, MON – месяц;
- YY, YYYY – год;
- HH24, HH24 – часы;
- Mi, mi –минуты;
- SS, ss – секунды;
- FF, ff – тики;
- разделители – тире «-», косая черта «/», двоеточие «:», точка «.» и др. знаки, в том числе символьные строки, не совпадающие с перечисленными выше обозначениями.

Примеры форматов даты

```
"DD-Mon-YY", "DD-Mon-YYYY"

"MM/DD/YY", "MM/DD/YYYY"

"DD.MM.YY", "DD.MM.YYYY"
```

Примеры форматов времени

```
"HH24"

"HH24:MI"

"HH24:MI:SS"

"HH24:MI:SS.FF"
```

Описание

Возвращается символьное представление <даты>:

- в формате DD.MM.YYYY, если параметр <формат представления> не задан или равен FALSE;
- в формате DD.MM.YYYY:HH:MI:SS.TT, если параметр <формат представления> равен TRUE;
- в заданном формате, если параметр <формат представления> является строковым литералом.

Примеры

```
cur_dat:=18.11.1997:14:27:48.89;
str_dat:=dtoa(cur_dat);          // '18.11.1997:14:27:48.89'
str_dat:=dtoa(cur_dat,FALSE);    // '18.11.1997'
str_dat:=dtoa(cur_dat,TRUE);     // '18.11.1997:14:27:48.89'
```

Преобразование байтового значения в строку

Синтаксис

btoa (<строка>[,<флаг>])

<строка> – значение типа BYTE.

Возвращаемое значение

Байтовое значение в виде строки, причем, если <флаг> не задан или равен FALSE, значения выдаются в виде идущих подряд шестнадцатеричных пар чисел (в соответствии со SQL-функцией hex), иначе значения разделяются пробелами.

Универсальное преобразование в строку

Синтаксис

tochar (<значение> [, <параметры>])

<значение> – выражение любого допустимого типа.

Описание

Выполняет универсальное преобразование любого типа в строку. Фактически, для числовых типов, дат и байтовых значений, эквивалентно функциям itoa, ftoa, ntoa, dtoa, btoa, вызванным в зависимости от типа <значения>, при этом <параметры> соответствуют дополнительным параметрам этих функций (например, формат для dtoa, точность для ntoa и т.д.).

Функция удобна тем, что она универсальна, не надо помнить, какого типа значение, главное, что получаем строку. Для NULL-значения возвращается строка «NULL», а для типа NCHAR выполняется перекодировка из UNICODE в рабочую кодировку символьного типа.

Преобразование строки в дату

Синтаксис

atod| to_date (<строка> [,<формат представления>]),

<строка> – выражение символьного типа.

<формат представления> – строковый литерал.

Описание

Возвращается значение типа DATE, полученное в результате преобразования параметра <строка>, который должен иметь символьное представление даты в соответствии с

<форматом представления>. Если параметр <формат представления> не задан, <строка> должна быть представлена в формате по умолчанию DD.MM.YYYY[:HH[:MI[:SS[.TT]]]].

Если параметр содержит неверное представление даты, возвращается 0.0.0:0:0:0.0

Возвращаемое значение

Тип возвращаемого значения – DATE;

При ошибке преобразования возвращается начальная дата.

Примеры

```
str_dat:="18.11.1997:14:27:48.89";
cur_dat:=atod(str_dat);        // 18.11.1997:14:27:48.89

str_dat:="18.11.1997:14:27:48";
cur_dat:=atod(str_dat);        // 18.11.1997:14:27:48.0

str_dat:="18.11.1997:14:27";
cur_dat:=atod(str_dat);        // 18.11.1997:14:27:0.0

str_dat:="18.11.1997:14";
cur_dat:=atod(str_dat);        // 18.11.1997:14:0:0.0

str_dat:="18.11.1997";
cur_dat:=atod(str_dat);        // 18.11.1997:0:0:0.0

str_dat:="18.11";
cur_dat:=atod(str_dat);        // 0.0.0:0:0:0.0

str_dat:="18";
cur_dat:=atod(str_dat);        // 0.0.0:0:0:0.0

str_dat:="";
cur_dat:=atod(str_dat);        // 0.0.0:0:0:0.0

str_dat:="18.15.1997:14:27:48.89";
cur_dat:=atod(str_dat);        // 0.0.0:0:0:0.0


dt:=to_date("28.04.2000","dd.mm.yyyy"); // 28.04.2000:00:00:00.00


dt:=to_date("01","mm"); //31.01.0001:00:00:00.00
```

Преобразование в тип smallint

Синтаксис

tosmallint (<значение>)

<значение> – выражение символьного или любого числового типа.

Описание

Возвращается значение типа smallint, полученное в результате преобразования параметра <значение> по следующим правилам:

- если параметр содержит дробную часть, то она отбрасывается;
- преобразование заканчивается при обнаружении нецифрового знака в символьной строке;

- если значение параметра выходит за пределы допустимого диапазона (от –32767 до +32767), то возвращается остаток от деления значения параметра на 65536 с учетом полученного знакового разряда (т.е. <значение> mod 65536).

Примеры

```
sml_int:=tosmallint("148");          // 148
sml_int:=tosmallint(148);            // 148
sml_int:=tosmallint("-34");          // -34
sml_int:=tosmallint(-34);            //-34
sml_int:=tosmallint("+34");          // 34
sml_int:=tosmallint(+34);            // 34
sml_int:=tosmallint(148-56/8+10);    //151
sml_int:=tosmallint("32767");        // 32767
sml_int:=tosmallint("65535");        // -1
sml_int:=tosmallint("65536");        // 0
sml_int:=tosmallint("70000");        // 4464
sml_int:=tosmallint("-70000");       // -4464
sml_int:=tosmallint("6fs65");        // 6
sml_int:=tosmallint("65.9");         // 65
sml_int:=tosmallint(65535*2+1000);   // 998
```

Функции данного класса преобразуют переданное значение в числовое значение соответствующего типа. При этом если передана строка, и она не является допустимой записью числа, возвращается 0.

Перобразование в тип int

Синтаксис

tointeger (<значение>)

<значение> – выражение символьного или любого числового типа.

Описание

Возвращается значение типа `int`, полученное в результате преобразования параметра <значение> по следующим правилам:

- если параметр содержит дробную часть, то она отбрасывается;
- преобразование заканчивается при обнаружении нецифрового знака в символьной строке;
- если значение параметра выходит за пределы допустимого диапазона (от –2 147 483 648 до +2 147 483 647), то возвращается остаток от деления значения параметра на 2 147 483 648*2 с учетом полученного знакового разряда (т.е. <значение> mod 2 147 483 648*2).

Преобразование в тип bigint

Синтаксис

tobigint (<значение>)

<значение> – выражение символьного или любого числового типа.

Описание

Возвращается значение типа bigint, полученное в результате преобразования параметра <значение> по следующим правилам:

- если параметр содержит дробную часть, то она отбрасывается;
- преобразование заканчивается при обнаружении нецифрового знака в символьной строке;
- если значение параметра выходит за верхний предел допустимого диапазона (+9 223 372 036 854 775 807), то возвращается значение верхнего предела, если же значение параметра выходит за нижний предел диапазона (–9 223 372 036 854 775 808), то возвращается значение нижнего предела.

Преобразование в тип real

Синтаксис

toreal (<значение>)

<значение> – выражение символьного или любого числового типа.

Описание

Возвращается значение типа double, полученное в результате преобразования параметра <значение> с учетом следующего правила:

- преобразование заканчивается при обнаружении нецифрового знака в символьной строке.

Преобразование в тип numeric

Синтаксис

tonumeric (<значение>)

<значение> – выражение символьного или любого числового типа.

Описание

Возвращается значение типа decimal, полученное в результате преобразования параметра <значение> с учетом следующего правила:

- преобразование заканчивается при обнаружении нецифрового знака в символьной строке.

Преобразование символьной строки в строку байт

Синтаксис

hextoraw (<символьное выражение>)

<символьное выражение> – строка типа BYTE, VARBYTE, содержащая символьное представление шестнадцатеричных цифр (цифры 0-9, буквы A-F).

Описание

Преобразование символьной шестнадцатеричной строки в строку байт.

Длина <символьного выражения> должна быть кратна 2.

Возвращаемое значение

Байтовая строка длиной N, если исходное <*символьное выражение*> имело длину 2*N.

Пример

```
var bt byte(10);
line:="4a3fbc09";
bt:=hextoraw(line); // 4a3fbc09000000000000
```

Преобразование значения в шестнадцатеричное представление

Синтаксис

rawtohex (<значимое выражение>)

<значимое выражение> – значение любого допустимого типа данных.

Описание

Преобразование значения в символьное шестнадцатеричное представление.

Возвращаемое значение

Символьная строка (тип CHAR) длиной 2*N, если исходное <*значимое выражение*> имело длину N.

Примеры

1)

```
i:=56;
line:=rawtohex(i); // 56
```

2)

```
line:=rawtohex("4a3fbc09"); // 3461336662633039
```

Функции для работы с курсорами

Проверка выхода курсора за пределы выборки

Синтаксис

outofcursor (<курсор>)

<курсор> – курсорная переменная.

Описание

Возвращается логическое значение TRUE, если была попытка выбрать ответ за пределами выборки, иначе – FALSE. Выбор за пределами происходит, если

выполняется FETCH NEXT на последнем ответе, FETCH PREVIOUS на первом или в результате FETCH ABSOLUTE/ FETCH RELATIVE, если осуществляется запрос на ответ с несуществующим номером.

Пример

```
//Типичная последовательность операторов для выборки всех ответов
open curs for ...;    // открыть курсор
fetch curs last;      // для выборки в обратном порядке
while not outofcursor(curs) loop
...
обработка ответа
...
fetch curs;          // fetch curs previous; для выборки в обратном порядке
Endloop
```

Количество ответов в курсорной выборке, сделанной по курсору

Синтаксис

rowcount (<курсор>)

<курсор> – курсорная переменная.

Описание

Возвращается значение типа INTEGER – количество ответов в выборке, сделанной по курсору.

Определение кода завершения для курсора

Синтаксис

errcode ([<курсор>])

<курсор> – имя курсорной переменной.

Описание

Возвращает значение типа INTEGER – код завершения СУБД ЛИНТЕР для данного курсора. Если курсорная переменная не задана, возвращается код завершения для последнего выполненного оператора EXECUTE.

Пример

```
...
Execute 'select * from AUTO where personid=345';
If ERRCODE() <> 0 THEN
return (TRUE)
ENDIF
...
...
EXCEPTIONS
  WHEN DIVZERO THEN
// использование ERRCODE в этом случае не имеет смысла
<операторы>
...
```

```
WHEN BADPARAM, BADRETVAL, UNDEFPROC THEN
print ('Ошибка определения процедуры:' errcode()
// здесь использование ERRCODE имеет смысл только для
тех кодов ошибок,
// которые не перечислены явно в объявлении исключений
EXCEPTIONS
  WHEN OTHERS THEN
  case errcode()
  ...
  <операторы>
  ...
```

Определение номера текущей строки курсора

Синтаксис

currow (<курсор>)

<курсор> – имя курсорной переменной.

Описание

Возвращает номер текущей строки открытого курсора или 0, если курсор не открыт

Транзакции в процедурах

Общие положения

Внутри хранимой процедуры поддерживается собственный механизм управления транзакциями:

- начало транзакции (begin transaction);
- окончание транзакции с подтверждением (commit transaction);
- окончание транзакции с откатом (rollback transaction).

Все транзакции в процедуре СУБД ЛИНТЕР начинает в том транзакционном режиме, который уже был установлен в канале (со снятым флагом AUTOCOMMIT для версии 6.1). Если же в процедурном канале не был установлен транзакционный режим, то в версии 6.1 в момент начала транзакции в процедурном канале снимается флаг AUTOCOMMIT и устанавливается режим READ COMMITED, а для версий 5.7, 5.9 и 6.0 устанавливается флаг EXCLUSIVE. При выходе из транзакционной секции («begin transaction» – «commit/rollback transaction») самого верхнего уровня восстанавливается исходный транзакционный режим.

Вложенные транзакции

Разрешается инициировать транзакцию внутри другой транзакции. Если процедура была запущена в режиме AUTOCOMMIT, то команда «commit transaction» приводит к фиксации изменений только тогда, когда выполняется команда «commit transaction» самой внешней транзакционной секции. Если же процедура была запущена в каком-то из транзакционных режимов (EXCLUSIVE, Optimistic и т. п.), то команда «commit transaction» вообще не приводит к фиксации изменений (они должны быть зафиксированы «извне» процедуры с помощью команды «commit»), а лишь отмечает конец секции «begin transaction». В отличие от «commit transaction», команда «rollback transaction» всегда приводит к откату транзакции до последней команды «begin transaction».

При подаче команды «begin transaction» автоматически создаётся особая контрольная точка в текущем канале процедуры. Соответственно, команда «rollback transaction» будет производить свою операцию rollback до этой контрольной точки, причём откат будет производиться и по дочерним каналам.

На этапе трансляции процедуры отслеживаются лишь экстремальные случаи несвязанных команд «rollback transaction», «commit transaction» и «begin transaction»: когда есть «begin transaction» и нет ни одной команды «commit/rollback transaction» и наоборот, когда есть команды «commit/rollback transaction» и нет ни одной «begin transaction». В этом случае выдаётся код завершения 10085 «Несоответствие числа команд begin transaction и commit/rollback transaction».

Остальные случаи несоответствия уровней вложенности транзакций отслеживаются уже на этапе выполнения процедуры. При обнаружении несоответствий уровней вложенности (попытка вызвать «commit/rollback transaction» на нулевом уровне вложенности или возвращение из процедуры с ненулевым уровнем вложенности) выдаётся исключение INVTRSTATE (идентификатор «-18») «Неверное состояние транзакции».

Пример вызова «commit transaction» на нулевом уровне вложенности:

```
code
  begin transaction;
  …
  rollback transaction;
  …
  commit transaction;
  return;
end;
```

Пример возвращения из процедуры с ненулевым уровнем вложенности:

```
code
  begin transaction;
  …
  begin transaction;
  …
  rollback transaction;
  return;
end;
```

Инициирование транзакции

Синтаксис

begin transaction

Описание

Функция увеличивает уровень вложенности транзакций в хранимой процедуре и устанавливает автоматически сгенерированную контрольную точку. Если процедура была запущена в режиме AUTOCOMMIT и вызов «begin transaction» – первый, то для версий СУБД ЛИНТЕР 5.7, 5.9 и 6.0 устанавливается режим EXCLUSIVE, а для версии 6.1 – сбрасывается флаг AUTOCOMMIT и, в случае отсутствия транзакционного режима, устанавливается режим READ COMMITED.

Подтверждение транзакции

Синтаксис

commit transaction

Описание

Реальное завершение транзакции выполняется только в том случае, если текущий уровень вложенности транзакций равен 0 и если процедура была запущена в режиме AUTOCOMMIT, в противном случае осуществляется только уменьшение уровня вложенности транзакций.

Откат транзакции

Синтаксис

rollback transaction

Описание

Откат транзакции выполняется до последней контрольной точки.

Пример

Последовательность запросов (утилита inl):

```
create or replace table test(i int);
create or replace procedure tr_test() for debug
code
  begin transaction; //
  execute direct "insert into test values (1);"; //
    begin transaction; //
    execute direct "insert into test values (2);"; //
    rollback transaction; //
  execute direct "insert into test values (3);"; //
  commit transaction; //
end;
execute tr_test();
select * from test;


I
-
|          1|
|          3|
```

Логические функции

Логическое “И”

Синтаксис

<число1> **&** <число2>

<число1> и <число2> – числовые выражения.

Описание

Возвращается результат битовой логической операции “И” из двух значений <число1> и <число2>.

Пример

```
declare var bi bigint;//
code
  bi := sysevent(); //
  if ( ( bi & 131072 ) > 0 ) then  print( "after" ); endif; //
…
```

Логическое "ИЛИ"

Синтаксис

<число1> | <число2>

<число1> и <число2> – числовые выражения.

Описание

Возвращается результат битовой логической операции "ИЛИ" из двух значений <число1> и <число2>.

Прочие функции

Нахождение максимального числа

Синтаксис

max (<число1>,<число2>),

<число1> и <число2> - числовые выражения.

Описание

Возвращается максимальное из двух значений <число1> и <число2>.

Примеры

```
max_num:=max(5,10);                              // 10

max_num:=max(5,8.6);                             // 8

max_num:=max(-5,-10);                            // -5

max_num:=max(tointeger("345"),toreal(567));      // 567

max_num:=max(5,max(10,max(4,9)));                // 10
```

Вычисление остатка от деления

Синтаксис

mod (<делимое>,<делитель>)

Описание

Вычисление остатка от деления.

Возвращаемое значение

Функция возвращает остаток от деления <делимого> на <делитель>, которые могут быть выражениями любого числовго типа.

Если значение <делителя> равно 0, возбуждается исключение BADPARAM.

Тип возвращаемого результата – DOUBLE.

Если какой-нибудь из аргументов NULL, результат NULL.

Генерация псевдослучайного числа

Синтаксис

rand ()

Описание

Получение псевдослучайного числа.

Возвращаемое значение

Функция возвращает положительное псевдослучайное целое число в диапазоне от 0 до 2147483647.

Тип возвращаемого результата – BIGINT.

Для получения различных последовательностей псевдослучайных чисел датчик случайных чисел можно инициализировать при помощи функции RANDOMIZE().

Инициализация датчика случайных чисел

Синтаксис

randomize (<значение>)

<значение> ::= <целое числовое выражение>

Описание

Инициализация датчика случайных чисел.

Возвращаемое значение

Функция инициализирует датчик случайных чисел заданным <значением> или согласно текущему значению системного таймера, если <значение> не задано, меньше либо равно нулю или равно NULL.

Функция возвращает значение, которым был инициализирован датчик.

Тип возвращаемого результата – INT.

Определение текущего пользователя

Синтаксис

username ()

Описание

Получение имени текущего пользователя.

Возвращаемое значение

Имя пользователя (строка), от лица которого выполняется транзакция.

Получение имени базы данных

Синтаксис

dbname ()

Описание

Функция предоставляет имя базы данных.

Возвращаемое значение

Имя БД, заданное при ее создании.

Пример

```
create or replace procedure TEST_PROC() result char(18)
code
return dbname(); //
end;
```

Преобразование строки по алгоритму md5

Синтаксис

encode_string (<результат>, <строка>[,<ключ>])

<результат> – результирующая строка типа BYTE

<строка> – преобразуемая строка типа CHAR

<ключ> – ключ преобразования. типа CHAR

Описание

Одностороннее преобразование строки по алгоритму md5 с заданным ключом.

Если параметр <ключ> не задан, используется ключ преобразования по умолчанию.

Функция позволяет создавать пользователей БД непосредственно в теле процедуры.

Возвращаемое значение

Результат преобразования длиной 16 байт возвращается в переменной <результат>. Если длина этой переменной меньше 16 байт, то заносятся только первые байты результата, если больше - оставшиеся байты заполняются нулями.

Пример

```
create table upwds(uname char(20), pwd byte(16));

drop procedure add_user;
create procedure add_user(in uname char(20); in pwd char(20))
for debug
declare
  var encstr byte(16); //
code
  encode_string(encstr, pwd); //
  execute "insert into upwds values(?,?);", uname, encstr; //
```

```
end;

drop procedure check_user;
create procedure check_user(in uname char(20); in pwd char(20)) result int
for debug
declare
  var encstr byte(16); //
code
  encode_string(encstr, pwd); //
  execute direct makestr("select * from upwds where uname='?' and
pwd=hex('?');",
          uname, encstr); //
  if errcode() = 2 then
     return 0; // not correct
  else
     return 1; //
  endif
end;
```

Генерация пользовательского кода завершения

Синтаксис

raise_error (<код >)

<код > – целочисленное значение из диапазона от 10200 до 10999 (включительно).

Описание

Функция завершает текущую исполняемую процедуру или триггер и выставляет указанный <код> в качестве кода завершения всего запроса, т.е. запроса execute на запуск процедуры или SQL–оператора, инициировавшего запуск триггера. Это позволяет, в частности, выполнить в триггере откат SQL–запроса, вернув соответствующий пользовательский код завершения.

Возвращаемое значение

Заданный код завершения.

Момент срабатывания триггера

Синтаксис

sysevent ()

Описание

Функция предоставляет маску событий, которые вызвали активизацию триггера в канале с идентификатором SESSIONID (идентификатор канала передается функции неявно). Функция должна использоваться внутри тела триггера.

Возвращаемое значение

Целочисленное значение типа INT, являющееся комбинацией масок возможных событий (определение масок событий содержатся в файле global.h)

При использовании в хранимой процедуре всегда возвращается 0.

Мнемоническое обозначение	Числовое значение	Причина активизации триггера
TRIG_INSERT	0x00000001	Выполнение операции INSERT
TRIG_UPDATE	0x00000002	Выполнение операции UPDATE
TRIG_UPDATE_OF	0x00000004	Выполнение операции UPDATE OF
TRIG_DELETE	0x00000008	Выполнение операции DELETE
TRIG_LOGON	0x00000010	Выполнение операции LOGON
TRIG_LOGOFF	0x00000020	Выполнение операции LOGOFF
TRIG_FOREACHROW	0x00001000	Вызов триггера для каждой строки
TRIG_FOREACHSTAT	0x00002000	Вызов триггера для всей таблицы
TRIG_BEFORE	0x00010000	Вызов триггера перед выполнением операции
TRIG_AFTER	0x00020000	Вызов триггера после выполнения операции
TRIG_INSTEAD	0x00040000	Вызов триггера вместо выполнения операции

Пример

```
create or replace table i (i int );
create or replace trigger ti before insert or update or delete on i for each
row execute
declare
  var bi bigint;//
code
  print( "begin -----------------------------------");//
  print( "sessionid into procedure = "+itoa(sessionid));//
  execute direct "select sessionid;" into bi;//
  print( "sessionid from query     = "+itoa(bi));//
  bi := sysevent(); //
  print( "sysevent = "+itoa(bi) );//
  if ( ( bi & 131072 ) > 0 ) then
    print( "after" );//
  else
    if ( ( bi & 65536 ) > 0 ) then
      print( "before" );//
    endif; //
  endif; //
  if ( ( bi & 16 ) > 0 ) then
    print( "logon" );//
  else
    if ( ( bi & 32 ) > 0 ) then
      print( "logoff" );//
    endif; //
  endif; //
  if ( ( bi & 1 ) > 0 ) then
    print( "insert" );//
  else
    if ( ( bi & 2 ) > 0 or ( bi & 4 ) > 0 ) then
      print( "update" );//
    else
      if ( ( bi & 8 ) > 0 ) then
        print( "delete" );//
      endif; //
    endif; //
  endif; //
  print( "end -------------------------------------");//
end;
insert into i values (1);
update i set i=2;
delete from i;
```

```
drop table i;
```

Приостанов выполнения процедуры

Синтаксис

sleep(<длительность>)

<длительность> – целочисленное положительное значение типа BIGINT.

Описание

Функция SLEEP предназначена для «усыпления» хранимой процедуры, в контексте которой она была вызвана. При этом управление передаётся другим работающим в данное время запросам (процедурам) канала. По прошествии промежутка времени, заданного аргументом функции, процедуре возвращается управление, и она продолжает свою работу (это не означает, что управление функции будет передано *немедленно* по прошествии этого интервала, управление будет передано на очередном кванте, выделенном СУБД для канала, в котором обрабатывается процедура).

Аргумент <длительность> задаёт количество миллисекунд, на которое процедура должна уснуть. Максимальное значение равно 0x3FFFFFFF (или 1073741823 в десятичной записи), что составляет приблизительно 12.4 суток.

Пример

С помощью утилиты inl создаём две процедуры, печатающие на консоль локально запущенного ядра СУБД ЛИНТЕР время до и после выполнения функции sleep, причём первая процедура «засыпает» на 1 секунду, а вторая – на 5 секунд:

```
create or replace procedure test_sleep1(in n int) for debug
declare
  var i int; //
code
  i := 0; //
  while i <= n loop
    print("--- Proc 1 start (1 sec): " + dtoa(SYSDATE())); //
    sleep(1000); //
    print("--- Proc 1 stop  (1 sec): " + dtoa(SYSDATE())); //
    i := i + 1; //
  endloop; //
end;

create or replace procedure test_sleep2(in n int) for debug
declare
  var i int; //
code
  i := 0; //
  while i <= n loop
    print("<<< Proc 2 start (5 sec): " + dtoa(SYSDATE())); //
    sleep(5000); //
    print(">>> Proc 2 stop  (5 sec): " + dtoa(SYSDATE())); //
    i := i + 1; //
  endloop; //
end;
```

Затем с помощью двух утилит inl (по возможности одновременно) выполняем следующие запросы:

1) выполнить 20 циклов по 1 секунде

```
execute test_sleep1(20);
```

2) выполнить 4 цикла по 5 секунд:

```
execute test_sleep2(4);
```

Из протокола выполнения видно, что процедуры действительно «засыпают» и «просыпаются» через заданный промежуток времени.

Предметный указатель

ABSOLUTE, 26, 88
AND, 32
APPENDACTIVE, 17
APPENDACTIVE, 34
APPENDNOTSTARTED, 17
APPENDNOTSTARTED, 35
AS, 12, 25
BADCURSOR, 17, 19, 26, 34
BADPARAM, 17, 19, 24, 31, 34, 72, 74, 91
BADRETVAL, 17, 19, 24
BIGINT, 8, 10
BOOL, 9
BYTE, 8
CHAR, 8
CODE, 18
CURNOTOPEN, 17, 19, 26, 27
CURSOR, 9, 12
CUSTOM, 17, 18
DATE, 9, 32
DECLARE, 16
DEFAULT, 15, 16, 17, 33
DELETING, 13
DIRECT, 29
DIVZERO, 17, 19, 30, 31
DOUBLE, 8, 46
EIF, 33
ELSE, 21, 33
ELSEIF, 21, 22
ENDCASE, 22
ENDIF, 21
ENDLOOP, 22
EXCEPTION, 17
EXCEPTIONS, 18
FALSE, 12, 21, 23, 88
FIRST, 26
FOR, 17, 25
FOR DEBUG, 15, 16
IGNORE, 18, 19
IN, 15, 24
INOUT, 15, 24
INSERTING, 13
INT, 8, 10, 46
INTEGER, 8, 10, 26, 46
INTO, 23, 24
LAST, 26
LOOP, 22
NCHAR, 9
NEW, 12
NEXT, 26, 88
NOT, 32
NULL, 12, 15, 17, 24, 25, 32, 33
NULLDATA, 12, 17
NUMERIC, 8, 46
NVARCHAR, 9
OLD, 12
OR, 32
OUT, 15, 24
PREVIOUS, 26, 88
PROCEDURE, 15
QUERYWHENAPPEND, 17, 35
REAL, 8, 46
RELATIVE, 26, 88
RELEASE, 28
RESULT, 9, 15, 16
ROWCOUNT, 14
SESSIONID, 14
SMALLINT, 8, 10, 26, 46
THEN, 21, 22
TRUE, 12, 21, 23, 33, 54, 65, 88
TYPEOF, 9
UNDEFPROC, 17, 19, 24
UPDATING, 13
VAR, 17
VARBYTE, 8
VARCHAR, 8
WHEN, 18, 19
WHEN OTHERS, 18, 19, 22
WHERE CURRENT, 25
запрос
  динамический, 29
  претранслируемый, 29
идентификатор, 7, 8
имя. *См.* идентификатор
ключевое слово, 7
комментарий, 8
константы
  DATE, 11
  UNICODE, 11
  логические, 12
  символьные, 10
  числовые, 10
лексема, 7
модификатор, 15
хранимая процедура, 15

Указатель операторов

CALL, 23, 25
CASE, 22
CLOSE, 27
COMMIT, 28
EXECUTE, 27, 37, 44, 89
FETCH, 25, 26, 27, 33, 37, 88
GOTO, 23
IF, 21
OPEN, 25, 26, 27
RESIGNAL, 17, 19, 25
RETURN, 24
ROLLBACK, 28
SIGNAL, 24
WHEN, 22
WHILE, 22, 26
выражение, 21

Указатель функций

&

&, 92

|

|, 92

A

abs, 68
acos, 71
add_blob, 37, 38
add_months, 77
asc, 58
asin, 71
atan, 71
atan2, 71
atod, 84

B

begin transaction, 91
blob_size, 37, 44
btoa, 83

C

ceil, 68
char_length, 50
chr, 58
clear_blob, 37
clearPutm, 35
commit transaction, 91
cos, 70
cosh, 71
currow, 89

D

date_round, 69, 70
datesplit, 78
day, 74
dbname, 94
difference, 62
divtime, 80
dtoa, 82
dupchar, 57

E

encode_string, 94
errcode, 89
exp, 72

F

floor, 69, 70
flushPutm, 35
ftoa, 81

G

getPutmRecs, 35

H

hextoraw, 87
hour, 75

I

initcap, 59
insert, 54, 59
instr, 49
int getPutmRecs, 35
itoa, 80

L

last_day, 76
len, 50
length, 50
ln, 72
log, 72
lower, 59
lpad, 46
ltrim, 52

M

make_date, 76
makestr, 57
max, 92
minute, 75
mod, 93
modify_blob_type, 44
month, 74
multime, 79

N

next_day, 76
ninitcap, 67
ninsert, 67
nlen, 63
nlower, 66
nlpad, 64
nltrim, 66
nrepeat_string, 65

nreplace, 67
nright_substr, 65
nrpad, 64
nrtrim, 67
nstrpos, 65
nsubstr, 64
ntoa, 82
ntolower, 66
ntoupper, 65
ntranslate, 68
ntrim, 63
nupper, 65

O

octet_length, 51
outofcursor, 88

R

raise_error, 95
rand, 93
randomize, 93
rawtohex, 87
read_blob, 37, 39
read_blob_bigint, 37, 40
read_blob_bool, 37, 43
read_blob_char, 37, 42
read_blob_date, 37, 43
read_blob_double, 37, 41
read_blob_int, 37, 40
read_blob_ncar, 42
read_blob_nchar, 37
read_blob_numeric, 37, 41
read_blob_real, 37, 41
read_blob_smallint, 37, 40
repeat_string, 49
replace, 55
right_substr, 48
rollback transaction, 91
round, 72
rowcount, 88
rpad, 47
rtrim, 52

S

second, 75
seek_blob, 37, 38, 43
set_cur_blob, 37, 44
sign, 73
sin, 70
sinh, 71
sleep, 97
soundex, 62
sqrt, 73
strpos, 54
substr, 53
sysdate, 76
sysevent, 96

T

tan, 70
tanh, 71
ticks, 75
to_char, 60
to_date, 84
tobigint, 86
tochar, 84
tointeger, 86
tolower, 59
tonchar, 66
tonumeric, 87
toreal, 86
tosmallint, 85
touper, 58
translate, 56
trim, 51
trunc, 73

U

upper, 58
username, 94

Y

year, 74

Анкета для читателей

Цель данной анкеты – сбор информации для создания документации, наиболее полно отвечающей требованиям наших клиентов. Пожалуйста, заполните ее и отправьте одним из способов:

- по почте: 394000, г. Воронеж, а/я 137.
- по факсу: (473) 2-711-711, 2-778-333.
- по электронной почте (e-mail): market@relex.ru.

Процедурный язык	
Вопрос	**Обведите нужный ответ**
Была ли информация достаточно полной?	Да / Нет
Легко ли было найти нужную информацию?	Да / Нет
Доступно ли изложена информация?	Да / Нет

Что Вам особенно понравилось (не понравилось) в организации, способе подачи информации, стиле и оформлении документа?

ФИО: ____________________

Должность: ____________________

Компания: ____________________

Адрес: ____________________

Телефон: ____________________

e-mail: ____________________

Для дополнительных заметок

На этом листе Вы можете записать свои замечания и все, что считаете необходимым сообщить сотрудникам компании РЕЛЭКС.