СИСТЕМА
УПРАВЛЕНИЯ
БАЗАМИ
ДАННЫХ

ЛИНТЕР®

ЛИНТЕР БАСТИОН

ЛИНТЕР СТАНДАРТ

Процедурный язык

НАУЧНО-ПРОИЗВОДСТВЕННОЕ ПРЕДПРИЯТИЕ

РЕЛЭКС

Товарные знаки

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

Интеллектуальная собственность

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

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

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

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

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

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

О документе

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

Контактные данные

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

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

e-mail: info@linter.ru.

Техническая поддержка

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

Содержание

Предисловие 6
Назначение документа 6
Для кого предназначен документ 6
Дополнительные документы 6
Общие сведения 7
Элементы языка 8
Лексемы 8
Литералы 9
Числовые литералы 10
Символьные литералы 10
UNICODE-литералы 11
Литералы типа DATE 11
Логические литералы 12
Курсорные литералы 12
NULL-значения 12
Имена (идентификаторы) 13
Типы данных 13
Хранимые процедуры 18
Создание хранимой процедуры 18
Модификация хранимой процедуры 21
Компиляция хранимой процедуры 21
Определение привилегий на хранимую процедуру 23
Отмена привилегий на хранимую процедуру 23
Выполнение хранимой процедуры 24
Удаление текста хранимой процедуры 27
Удаление хранимой процедуры 27
Временные процедуры 29
Выполнение временной процедуры 29
Предопределенные переменные триггера 31
Старое и новое значение поля в триггере 31
Триггерные предикаты 32
Количество обработанных строк 33
Идентификатор канала 33
Общий вид процедурного блока 35
Формат блока описаний 35
Формат блока кода 37
Формат блока обработки исключений 41
Глобальные переменные хранимых процедур 45
Операторы 48
Оператор-выражение 48
Условный оператор 48
Оператор выбора 49
Операторы цикла 50
Оператор цикла с предусловием 50
Оператор цикла с постусловием 51
Оператор цикла с параметром 52
Досрочное завершение текущего цикла 53
Досрочный переход к следующей итерации цикла 53
Безусловный переход 53
Возврат из процедуры 54
Вызов исключения 54
Передача исключения 54
Открытие курсора 55

Выборка данных из открытого курсора ... 56
Закрытие курсора ... 58
Выполнение запроса ... 58
Завершение транзакции ... 62
Выражения ... 64
Операнды ... 64
Операции ... 65
Операции в числовых выражениях ... 65
Операции в символьных выражениях ... 65
Операции в логических выражениях ... 66
Побитовые логические операции ... 66
Логическое «И» ... 66
Логическое «ИЛИ» ... 67
Операции в выражениях типа «дата» ... 67
Присвоение значений ... 69
Условные выражения ... 70
Пакетное добавление ... 71
Процедуры с курсорным параметром ... 76
Работа с типом данных BLOB ... 78
Установка текущего BLOB-столбца ... 79
Добавление данных в конец BLOB-значения ... 79
Чтение данных из BLOB-значения на общем уровне ... 81
Чтение данных типа int из BLOB-значения ... 81
Чтение данных типа smallint из BLOB-значения ... 81
Чтение данных типа bigint из BLOB-значения ... 82
Чтение данных типа real из BLOB-значения ... 82
Чтение данных типа numeric из BLOB-значения ... 83
Чтение данных типа DOUBLE из BLOB-значения ... 83
Чтение данных типа CHAR из BLOB-значения ... 83
Чтение данных типа NCHAR из BLOB-значения ... 84
Чтение данных типа DATE из BLOB-значения ... 84
Чтение данных типа bool из BLOB-значения ... 85
Установка текущей позиции для чтения BLOB-данных ... 85
Получение размера BLOB-значения ... 85
Удаление BLOB-значения ... 86
Типизация BLOB-значения ... 87
Длина BLOB-значения ... 88
Работа с пакетом данных BSON-формата ... 90
Доступ к документам пакета ... 91
Доступ к полям документа ... 92
Перемещение по документам пакета ... 93
Извлечение информации из поля документа ... 94
Извлечение информации из вложенного документа ... 96
Поддержка кодовых страниц ... 98
Функции ... 99
Стандартные функции ... 99
Символьные функции ... 99
Дополнение строки слева ... 99
Дополнение строки справа ... 100
Получение подстроки ... 101
Получение правой части строки ... 101
Дублирование строки ... 102
Поиск подстроки ... 103
Длина символьной строки ... 104
Длина байтовой строки ... 104

Удаление крайних пробелов из символьной строки 105
Удаление левосторонних символов 105
Удаление правосторонних символов 106
Поиск подстроки 107
Корректировка подстроки 107
Замена всех подстрок 109
Замена символов строки 110
Преобразование строки 110
Удвоение символа в строке 112
Преобразование байтовой строки в символьную 112
Преобразование символьной строки в байтовую 112
Числовое представление символа 113
Преобразование строки к верхнему регистру 113
Преобразование строки к нижнему регистру 113
Перевод начальной буквы слова в заглавную 114
Преобразование числового выражения в символьный вид 114
Фонетический код строки 116
Сравнение фонетического звучания строк 117
Символьные UNICODE-функции 117
Определение длины UNICODE-строки 117
Удаление пробелов UNICODE-строки 118
Выделение UNICODE-подстроки 118
Левостороннее дополнение UNICODE-строки 118
Правостороннее дополнение UNICODE-строки 119
Получение правосторонней подстроки UNICODE-строки 119
Дублирование UNICODE-строки 119
Поиск UNICODE-подстроки 120
Преобразование UNICODE-строки к верхнему регистру 120
Преобразование UNICODE-строки к нижнему регистру 120
Преобразование значения в UNICODE-строку 120
Удаление левосторонних символов из UNICODE-строки 121
Удаление правосторонних символов из UNICODE-строки 121
Перевод начальной буквы UNICODE-слова в заглавную 121
Корректировка UNICODE-подстроки 122
Замена всех UNICODE-подстрок 122
Замена символов UNICODE-строки 122
Математические функции 123
Вычисление абсолютного значения 123
Округление до целого с избытком 123
Округление до целого с недостатком 123
Округление значения типа «дата-время» 124
Усечение представления значения типа «дата-время» 125
Тригонометрические функции 126
Обратные тригонометрические функции 126
Гиперболические функции 127
Экспоненциальная функция 127
Логарифмические функции 127
Округление с заданной точностью 128
Усечение числа с заданной точностью 128
Определение знака числа 129
Вычисление квадратного корня числа 129
Календарные функции 129
Выделение дня из даты 129
Выделение месяца из даты 130
Выделение года из даты 130

Выделение часа из даты ... 131
Выделение минут из даты ... 131
Выделение секунд из даты ... 131
Выделение тиков из даты ... 132
Формирование даты ... 132
Получение текущей даты по Гринвичу ... 132
Получение текущей локальной даты ... 133
Последний день месяца ... 133
Дата очередного дня недели ... 133
Помесячное изменение даты ... 134
Выделение заданных элементов даты ... 135
Изменение даты на заданный интервал времени ... 136
Вычисление интервала между двумя датами ... 137
Функции преобразования типов ... 137
Представление числа в символьном виде ... 137
Представление числа в символьном виде с учетом знака ... 138
Представление числа в символьном виде с учетом знака и заданной точности ... 139
Представление даты в символьном виде ... 139
Преобразование байтового значения в строку ... 141
Универсальное преобразование в строку ... 141
Преобразование строки в дату ... 141
Преобразование в тип smallint ... 143
Преобразование в тип int ... 144
Преобразование в тип bigint ... 144
Преобразование в тип real ... 144
Преобразование в тип numeric ... 145
Преобразование символьной строки в строку байт ... 145
Преобразование значения в шестнадцатеричное представление ... 145
Функции для работы с курсорами ... 146
Проверка выхода курсора за пределы выборки ... 146
Количество записей в курсорной выборке, сделанной по курсору ... 146
Определение кода завершения SQL-команды ... 147
Определение сообщения завершения SQL-команды ... 147
Определение кода исключения ... 148
Определение номера текущей строки курсора ... 149
Прочие функции ... 149
Вычисление максимального значения из пары значений ... 149
Вычисление максимального значения из набора значений ... 150
Вычисление минимального значения из пары значений ... 151
Вычисление минимального значения из набора значений ... 152
Вычисление остатка от деления ... 153
Генерация псевдослучайного числа ... 153
Инициализация датчика случайных чисел ... 153
Определение имени текущего пользователя ... 154
Определение идентификатора текущего пользователя ... 155
Определение имени фактического исполнителя процедуры ... 155
Определение идентификатора фактического исполнителя процедуры ... 157
Получение имени базы данных ... 157
Преобразование строки по алгоритму md5 ... 157
Генерация пользовательского кода завершения ... 159
Момент срабатывания триггера ... 160
Приостанов выполнения процедуры ... 162
Одновариантная замена NULL-значения реальным значением ... 164
Двухвариантная замена NULL-значения реальным значением ... 165
Вывод сообщения ... 166

Транзакции в процедурах 167
Общие положения 167
Вложенные транзакции 167
Инициирование транзакции 168
Подтверждение транзакции 168
Откат транзакции 168
Приложение. Ключевые слова процедурного языка 170
Предметный указатель 171
Указатель операторов 173
Указатель функций 174

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

Для кого предназначен документ

Документ предназначен для программистов, разрабатывающих приложения, использующие триггеры и хранимые процедуры СУБД ЛИНТЕР.

Дополнительные документы

- Архитектура СУБД
- Справочник кодов завершения
- Справочник по SQL
- Запуск и останов СУБД ЛИНТЕР в среде ОС Windows
- Запуск и останов СУБД ЛИНТЕР в среде ОС Linux

© Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Общие сведения

Процедурный язык предназначен для написания текстов триггеров и хранимых процедур. Также в процедурном языке СУБД ЛИНТЕР реализована работа с глобальными переменными.

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

Лексемы

Определение лексических единиц процедурного языка.

Спецификация

[1] <лексема>::=
<литерал>
| <идентификатор>
| <ключевое слово>
| <разделитель>
[2] <идентификатор>::= <стандартный идентификатор>
[3] <стандартный идентификатор>::=
{<латинская буква> | $ } [{<цифра> | <латинская буква> | _ | $}…]
[4] <ключевое слово>::=
неопределяемый элемент синтаксической конструкции
[5] <разделитель>::=
, | (|) | < | > | . | : | = | * | + | - | / | <> | >= | <= | <комментарий>
[6] <комментарий>::=
{<однострочный комментарий> | <многострочный комментарий>}
[7] <однострочный комментарий>::= // [<символ> …]
[8] <многострочный комментарий>::= /* [<символ> …] */
[9] <символ>::= любой символ или графический знак

Синтаксические правила

1) <Идентификатор> может быть представлен двумя способами:

- без двойных кавычек. В этом случае символами <идентификатора> могут быть только цифры, латинские буквы, знаки подчеркивания (_) и доллара ($). <Идентификатор> может начинаться с символа $ или _ и не должен совпадать с ключевыми словами СУБД ЛИНТЕР. Все такие <идентификаторы> приводятся к верхнему регистру.

Примечание
Во избежание возможных коллизий не рекомендуется использовать имена, начинающиеся с символа $, так как их присваивает СУБД ЛИНТЕР создаваемым ею объектам БД.

Одинаковые идентификаторы:

```
BANK, Bank
```

- заключен в двойные кавычки. В этом случае могут использоваться любые символы. <Идентификатор> может совпадать с ключевыми словами СУБД ЛИНТЕР, и преобразование регистра представления знаков <идентификатора> не выполняется.

Разные идентификаторы:

```
"BANK", "Bank", "банк", "Банк"
```

Допустимые идентификаторы:

```
"Table", SYSTEM."Склад", "Табельный номер"
```


© Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

2) <Идентификаторы>, не упомянутые как ключевые в стандарте SQL, считаются ключевыми только в том случае, если они встречаются в определенном контексте.

   Таким образом, контекстно-зависимые ключевые слова распознаются только в том контексте, где они могут встретиться, в других случаях они считаются идентификаторами.

Литералы

[1] <литерал> : : =
<символьный литерал>
| <байтовый литерал>
| <двоичный литерал>
| <UNICODE-литерал>
| <числовой литерал>
| <дата-время литерал>
| <логический литерал>

[2] <символьный литерал> : : = ' [<представление символа>...] '

[3] <представление символа> : : =
цифра | буква | специальный символ | одинарная кавычка

[4] <одинарная кавычка> : : = ' '

[5] <байтовый литерал> : : =
<шестнадцатеричный байтовый литерал>
<шестнадцатеричный числовой литерал>

[6] <шестнадцатеричный байтовый литерал> : : =
HEX ('<представление байта>...')
| x'шестнадцатеричная цифра…'

[7] <шестнадцатеричный числовой литерал> : : =
[[+] | -]0xшестнадцатеричная цифра…

[8] <представление байта> : : =
шестнадцатеричная цифра[шестнадцатеричная цифра]

[9] <двоичный литерал> : : =
<двоичный байтовый литерал> |
<двоичный числовой литерал>

[10] <двоичный байтовый литерал> : : =
[[+] | -] b'<двоичная цифра>…'

[11] <двоичный числовой литерал> : : = [[+] | -] 0b<двоичная цифра>…

[12] <двоичная цифра> : : = 0 | 1

[13] <UNICODE-литерал> : : = N '<символьный литерал>'

[14] <числовой литерал> : : =
<целочисленный литерал>
| <точный числовой литерал>
| <приближенный числовой литерал>

[15] <целочисленный литерал> : : = [[+] | -] <беззнаковое целое>

[16] <точный числовой литерал> : : =
[[+] | -] <беззнаковое целое>.<беззнаковое целое>

[17] <приближенный числовой литерал> : : =
<мантисса> [E] <экспонента>

[18] <мантисса> : : = <точный числовой литерал>

[19] <экспонента> : : = [[+] | -] <беззнаковое целое>

[20] <беззнаковое целое> : : = цифра[цифра…]

[21] <дата-время литерал> : : = <представление даты>

[22] <представление даты> : : =
<формат 1> | <формат 2> | <формат 3> | <формат 4> | <формат 5> | <формат 6>

[23] <формат 1> : : = DD-MM-[YY]YY[:HH[:MI[:SS[.FF]]]]

[24] <формат 2> : : = MM/DD/[YY]YY[:HH[:MI[:SS[.FF]]]]

[25] <формат 3>::= DD.MM.[YY]YY[:HH[:MI[:SS[.FF]]]]
[26] <формат 4>::= DD-MON-[YY]YY[:HH[:MI[:SS[.FF]]]]
[27] <формат 5>::= YYYY-MM-DD[:HH[:MI[:SS[.FF]]]]
[28] <формат 6>::= YYYYMMDD
[29] <логический литерал>::=
TRUE | true | 'TRUE' | 'true' | FALSE | false | 'FALSE' | 'false'

Числовые литералы

Литералы 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).

Литералы целого типа по умолчанию приводятся к следующим типам:

Значение литерала	Тип по умолчанию
от -32 767 до +32 767	SMALLINT
от -2 147 483 647 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	

Литералы с десятичной точкой по умолчанию приводятся к типу `NUMERIC`.

Для указания конкретного типа, к которому должен быть приведен литерал, применяется суффикс, который добавляется в конец литерала.

В таблице 1 приведен перечень суффиксов и приводимых типов числовых литералов.

Таблица 1. Перечень суффиксов и приводимых типов числовых литералов

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

Примеры числовых литералов

```
0, +123I, -456, 3.1415, -0.33R, 777B
```

Символьные литералы

Литералы 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;

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

- \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 «\».

В длинных текстовых литералах, размещаемых на отдельных строках текста, символ перехода на следующую строку (EOL) сохраняется, то есть конструкция типа:

```
"aaa
bbb
ccc"
```

интерпретируется как "aaa\nbbb\nccc" символ \r удаляется, а в конструкции типа:

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

интерпретируется как "aaabbbccc".

Примеры символьных литералов

```
"abcd", "New Line\n", "\x7 the Bell", "\"c:\\linter\""
```

UNICODE-литералы

Чтобы задать литерал, представленный как UNICODE-значение, необходимо использовать обычный символьный литерал, который преобразуется к типу NCHAR при помощи функции TONCHAR. При этом кодировка символьного литерала соответствует кодировке, в которой работало клиентское приложение на момент создания процедуры (триггера), и преобразование в UNICODE будет выполняться именно с учетом этой кодировки.

Литералы типа DATE

Литералы типа DATE имеют формат DD.MM.YYYY[:HH[:MI[:SS[.FF]]]]. Это означает, что любая конечная последовательность, то есть :HH:MM:SS.FF, :MM:SS.FF, :SS.FF или .FF, 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;
- FF – тики (сотые доли секунд).

Примеры литералов типа DATE

```
21.08.1997
```

```
21.8.1997:20:53

21.08.1997:9:10.55
```

Логические литералы

Литералы типа BOOL – это ключевые слова TRUE и FALSE.

Курсорные литералы

Литералов 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. раздел Присвоение значений).

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

Для указания NULL-значения используется литерал NULL.

Если исключение NULLDATA не прописано в блоке EXCEPTION, то оно игнорируется.

Примеры

```
1) При выполнении эта процедура завершится с исключением:
CREATE OR REPLACE PROCEDURE tst_code() RESULT INT FOR DEBUG
DECLARE
  VAR v,z INT;
  EXCEPTION NULLDATA for NULLDATA;
CODE
  v := NULL;
  z := 1+v;
  RETURN z;
END;

2) А эта процедура будет выполняться без исключения:
CREATE OR REPLACE PROCEDURE tst_code() RESULT INT FOR DEBUG
DECLARE
  VAR v,z INT;
CODE
  v := NULL;
  z:=1+v;
  RETURN z;
END;
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Имена (идентификаторы)

Имена (идентификаторы) используются для именования передаваемых параметров, локальных переменных, исключений, меток исполняемых операторов.

Спецификация

[1] <имя схемы>::= <идентификатор>
[2] <имя таблицы>::= <идентификатор>
[3] <имя представления>::= <идентификатор>
[4] <имя столбца>::= <идентификатор>
[5] <имя хранимой процедуры>::= <идентификатор>
[6] <имя параметра>::= <идентификатор>
[7] <имя пользователя>::= <идентификатор>
[8] <имя роли>::= <идентификатор>
[9] <имя переменной>::= <идентификатор>
[10] <имя глобальной переменной>::= <идентификатор>
[11] <имя поля>::= <идентификатор>
[12] <имя исключения>::= <идентификатор>

Синтаксические правила

1) Длина <имя схемы>, <имя хранимой процедуры>, <имя пользователя>, <имя таблицы>, <имя столбца> не более 66 символов.
2) <Имя параметра>, <имя переменной>, <имя поля> и <имя исключения> не должны начинаться с цифры и могут содержать до 30 символов.
3) Если <имя схемы> не указано, то используется схема пользователя, от имени которого подан запрос.
4) Два полных имени таблицы равны только тогда, когда совпадают их <имя схемы> и <имя таблицы>.
5) Имя может состоять из последовательности любых алфавитно-цифровых и графических символов, за исключением следующих:

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

6) В качестве идентификатора нельзя использовать ключевые слова процедурного языка (см. приложение).

Типы данных

В процедурном языке СУБД ЛИНТЕР поддерживаются простой и составной типы данных.

Определение типов данных.

Спецификация

[1] <тип данных>::=
<простой тип данных>|<составной тип данных>|<тип данных объекта>
[2] <простой тип данных>::=
<строковый тип>
| <байтовый тип>
| <UNICODE тип>
| <точный числовой тип>
| <приближенный числовой тип>
| <дата-время тип>

| <логический тип>
| <BLOB-тип>

[3] <строковый тип>::=
<строковый тип фиксированной длины>
| <строковый тип переменной длины>

[4] <строковый тип фиксированной длины>::= CHAR [(<длина>)]

[5] <строковый тип переменной длины>::= VARCHAR (<длина>)

[6] <байтовый тип>::=
<байтовый тип фиксированной длины>
| <байтовый тип переменной длины>

[7] <байтовый тип фиксированной длины>::= BYTE [(<длина>)]

[8] <байтовый тип переменной длины>::= VARBYTE (<длина>)

[9] <UNICODE тип>::=
<UNICODE тип фиксированной длины>
| <UNICODE тип переменной длины>

[10] <UNICODE тип фиксированной длины>::= NCHAR [(<длина>)]

[11] <UNICODE тип переменной длины>::= NVARCHAR (<длина>)

[12] <точный числовой тип>::=
NUMERIC [(<точность> [,<масштаб>])]
| BIGINT
| {INTEGER | INT}
| SMALLINT

[13] <приближенный числовой тип>::= {REAL | DOUBLE}

[14] <дата-время тип>::= DATE

[15] <логический тип>::= BOOL

[16] <BLOB-тип>::= BLOB

[17] <длина>::= <беззнаковое целое>

[18] <точность>::= <беззнаковое целое>

[19] <масштаб>::= <беззнаковое целое>

[20] <составной тип данных>::= <курсор>|<массив>

[21] <курсор>::= CURSOR (<имя поля> <простой тип данных>[; …])

[22] <массив>::= ARRAY <простой тип данных>

[23] <тип данных объекта>::=
TYPEOF({<имя глобальной переменной>|[<имя схемы>.]{<имя таблицы>|<имя представления>}[.<имя столбца>]})

Синтаксические правила

Простой тип данных

1) `BIGINT` – используется для представления целых знаковых чисел в диапазоне от -9 223 372 036 854 775 807 до +9 223 372 036 854 775 807.
2) `INTEGER (INT)` – используется для представления целых знаковых чисел в диапазоне от -2 147 483 647 до +2 147 483 647.
3) `SMALLINT` – используется для представления целых знаковых чисел в диапазоне от -32 767 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 (целое или дробное) с максимальным масштабом (количество цифр справа от десятичной точки), равным 10, и точностью (максимальное число значащих цифр), равной 30.

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

7) CHAR(<размер>) – используется для представления алфавитно-цифровых символов, графических знаков и управляющих кодов в виде текстовой строки фиксированной длины (до 3910 знаков).

Примечание
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 3910 знаков).
9) BYTE(<размер>) – используется для представления алфавитно-цифровых символов, графических знаков и управляющих кодов в виде шестнадцатеричной строки фиксированной длины (до 3910 знаков).
10) VARBYTE(<размер>) – используется для представления алфавитно-цифровых символов, графических знаков и управляющих кодов в виде шестнадцатеричной строки переменной длины (до 3910 знаков).
11) NCHAR(<размер>) – используется для представления UNICODE-строки фиксированной длины (до 1955 знаков).
12) NVARCHAR(<размер>) – используется для представления UNICODE-строки переменной длины (до 1955 знаков).

Примечание
Длина строковых и байтовых типов данных может меняться от версии к версии – как в сторону уменьшения, так и в сторону увеличения.

13) DATE – используется для представления информации о дате и времени.

    Информация задается в формате:

    ```
    DD.MM.YYYY[:HH[:MI[:SS[.FF]]]].
    ```

    В таблице 2 приведены значения формата представления информации о дате и времени.

Таблица 2. Значения формат представления информации о дате и времени

Маска	Единицы	Диапазон
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)
FF	тики	(от 0 до 99)

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

14) BOOL – используется для представления логической информации. Допустимыми значениями являются TRUE и FALSE.

15) BLOB-значения – неструктурированные значения большого объема (Binary Large OBject). Максимальный размер BLOB-переменной составляет 2 Гбайта.

16) Работа с BLOB-значениями выполняется с помощью встроенных функций процедурного языка (см. раздел Работа с типом данных BLOB).

Примечание
С точки зрения процедурного языка тип данных BLOB является типом данных varbyte.

Составной тип данных

1) Тип данных CURSOR – используется для задания структуры данных, соответствующей полям таблицы или представления базы данных.
2) В процедурном языке поддерживаются одномерные безразмерные массивы.
3) Совместимость типов данных: для символьных выражений, дат и логических типов данных совместимы только сами эти типы. Для числовых выражений совместимы любые числовые типы.

Тип данных объекта

1) Если в конструкции TYPEOF() <имя объекта> не заключено в двойные апострофы, то оно будет приведено к верхнему регистру. Заключенное в двойные апострофы будет передано без изменения регистра. Допустимо с помощью двойных апострофов указывать запрещенные символы в имени объекта. Например, TYPEOF("$$$USR.$$$S34").
2) Правило определения объекта в конструкции TYPEOF():
   - если задано полное квалификационное <имя схемы>.<имя объекта>.<имя столбца> (например, TYPEOF(SYSTEM.AUTO.PERSONID)), то TYPEOF() применяется к указанному объекту.
   - если квалификационное имя задано не полностью (например, TYPEOF(a.b)), то:
     - предполагается вариант <имя объекта>.<имя столбца> для текущего пользователя;
     - предполагается вариант <имя схемы>.<имя объекта>.
   - если в конструкции TYPEOF() <имя объекта> задано без символов-разделителей (то есть без точек), то оно ищется в следующей очередности:
     - среди глобальных переменных процедурного языка;
     - среди имен базовых таблиц текущего пользователя;
     - среди имен представлений текущего пользователя.
3) Если <имя объекта> является именем столбца, то будет применен соответствующий простой тип данных.
4) Если <имя объекта> является <именем таблицы> или <именем представления>, то будет создана структура данных, соответствующая структуре записи объекта.

Примеры

```
1)
typeof(SYSTEM.AUTO.NAME);
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
2)
typeof("Склад"."Код_товара");

3)
create or replace table aaa (i int, ch char(3));
insert into aaa values(1, 'abc');
insert into aaa values(2, 'def');
insert into aaa values(3, 'ghi');

create or replace procedure prc_test01() result int 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 "select i, i, i from aaa;"; //
  l := c.k; //
  fetch c into i, j, k; //
  return(i + j + l + k); //
end;
call prc_test01();
Результат 7

4)
create or replace procedure prc_test02(in n1 int) result char(30)
declare
  var make typeof(auto.make); //
code
  execute "select make from auto where personid= ?" using n1 into
 make; //
  return "Модель " + make; //
end;
call prc_test02(1);
Результат Модель FORD
```

Хранимые процедуры

Создание хранимой процедуры

Определение оператора создания хранимой процедуры.

Спецификация

[1] <создание хранимой процедуры>::=
CREATE [IF NOT EXISTS | OR REPLACE] PROCEDURE
[<имя схемы>.]<имя хранимой процедуры>([<параметр> [; …]])
[AUTHID {CURRENT_USER | DEFINER}]
[RESULT {<простой тип данных> | <курсор> | <тип данных объекта>}] [FOR DEBUG]
<процедурный блок>

[2] <параметр>::=
<модификатор> <имя параметра> {<простой тип данных> | <курсор> | <тип данных объекта>}
[DEFAULT <инициализатор>]

[3] <модификатор>::= {IN | OUT | INOUT}

[4] <инициализатор>::= <литерал>

Синтаксические правила

1) Опция `IF NOT EXISTS` отменяет выполнение оператора, если указанная хранимая процедура уже существует в БД.
2) Опция `OR REPLACE` заставляет удалять существующую в БД хранимую процедуру и создавать её под тем же именем, но с указанными параметрами.
3) Одновременное использование опций `IF NOT EXISTS` и `OR REPLACE` запрещено.
4) Если <имя схемы> не указано, то процедура будет создана в текущей схеме.
5) Допустимы следующие значения <модификатора> параметра:
   - `IN` – передается в процедуру (используется только как входной параметр);
   - `OUT` – возвращается процедурой (используется только как выходной параметр);
   - `INOUT` – передается процедуре и возвращается ею (используется как входной/выходной параметр).
6) Тип параметра `CURSOR` можно использовать только при вызове процедуры внутри процедуры.
7) Список параметров может быть пустым или содержать до 128 параметров.
8) Допускается при указании однотипных параметров перечислять их через запятую в общей спецификации параметра, например,

```
create procedure prc_test (in v1, v2, v3 int; in c1, c2 char(10))
 result int …
```

9) При указании списка однотипных параметров допускается указывать соответствующий им список инициализаторов, задаваемый в виде списка выражений, разделенных запятыми.

```
create procedure prc_test (in v1, v2, v3 int default 1, abs(-2),
 3) result int …

create or replace procedure SUN (
  in v varbyte(10) default hextoraw("AABBCCDD");
```


© Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
  out v1 varbyte(10)) result varbyte(10) for debug
code
  v1:=v; //
  return v1; //
end;

call sun();
return AABBCCDD

create or replace procedure SUN (
  in nch nchar(10) default tonchar("alpha");
  out v1 nchar(10)) result nchar(10) for debug
code
  v1:=nch;
  return v1;
end;

call sun();
return alpha
```

10) Количество выражений в <инициализаторе> может быть меньше количества параметров в списке параметров. В этом случае параметры, для которых нет соответствующих выражений в <инициализаторе>, инициализируются так же, как при отсутствии фразы DEFAULT, то есть NULL-значениями (отсутствие DEFAULT равносильно явной записи DEFAULT NULL).

11) <Инициализатор> представляет выражение, которое может быть вычислено на этапе трансляции процедуры, то есть должно содержать константы/константные функции и ранее определенные в процедуре параметры и/или переменные (в последнем случае в качестве значений переменных берутся их значения по умолчанию).

12) При указании опции AUTHID DEFINER доступ к объектам, используемым в коде процедуры, осуществляется от имени владельца процедуры. При указании опции AUTHID CURRENT_USER проверяется доступ вызывающего процедуру пользователя к объектам, используемым в коде процедуры.

    При вызове процедуры, созданной с опцией AUTHID DEFINER, пользователю необходима привилегия EXECUTE AS OWNER на вызываемую процедуру. При вызове процедуры, созданной с опцией AUTHID CURRENT_USER, пользователю необходима привилегия EXECUTE на вызываемую процедуру.

Пример

```
drop user USER1 cascade;
create user USER1 identified by '12345678';
grant dba to USER1;
create user USER2 identified by '12345678';
grant dba to USER2;

username USER1/12345678
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 "INSERT INTO USER1.TEST values(?, ?)" using 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 "select ID from USER1.TEST where ID = ?;" using 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 USER2/12345678
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 оба запроса возвращают код завершения `1022` (нарушение привилегий).

13) 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`.
14) Если указана опция `FOR DEBUG`, процедура транслируется с отладочной информацией, иначе отладочная информация не включается в оттранслированный код процедуры, и процедуру нельзя будет отлаживать отладчиком хранимых процедур.
15) Опция `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
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Общие правила

1) Создавать процедуру от имени другого пользователя недопустимо.
2) При трансляции некорректной процедуры ее исходный код будет сохранен или перезаписан в БД для возможной последующей правки.

Модификация хранимой процедуры

Определение оператора модификации хранимой процедуры.

Спецификация

[1] <модификация хранимой процедуры>::=
ALTER PROCEDURE [<имя схемы>.]<имя хранимой процедуры>([<параметр> [; …]])
[AUTHID {CURRENT_USER | DEFINER}]
[RESULT {<простой тип данных> | <курсор> | <тип данных объекта>}] [FOR DEBUG]
<процедурный блок>

Синтаксические правила

1) Синтаксические правила аналогичны правилам формирования оператора Создание хранимой процедуры.

Общие правила

1) Выполнение команды допустимо для существующей в БД хранимой процедуры только ее владельцу.

Компиляция хранимой процедуры

Определение оператора компиляции существующей хранимой процедуры.

Спецификация

[1] <компиляция хранимой процедуры>::=
REBUILD PROCEDURE [<имя схемы>.]<имя хранимой процедуры>

Синтаксические правила

1) <Имя хранимой процедуры> должно ссылаться на существующую в БД хранимую процедуру.

Общие правила

1) Необходимость в перекомпиляции хранимой процедуры возникает в случае, если процедура использует претранслированные запросы, а указанные в них таблицы были пересозданы или изменена их структура.
2) Команда доступна только владельцу процедуры.
3) Для выполнения команды надо иметь привилегию RESOURCE.

Пример

```
grant resource to U1 identified by '12345678';
grant resource to U2 identified by '12345678';
username U1/12345678
create or replace table test (i int);
insert into test values (1);
insert into test values (2);
```

```
create or replace procedure cnt() result int
declare
  var a cursor(i int); //
code
  open a for "select count(*) from test;"; //
  return a.i; //
end;

! Должно быть  2
execute cnt();
grant execute on cnt to U2;
grant select on test to U2;
username U2/12345678
create or replace table test (i int);
insert into test values (1);
! Тоже должно быть 2

execute u1.cnt();
username U1/12345678
drop table test;
create table test(i int, k int);
insert into test values (1,1);
insert into test values (2,2);
insert into test values (3,3);
! Должно быть NULL -  таблица изменена
execute cnt();
grant select on test to U2;

username U2/12345678
! Команда не доступна   (выполняет не владелец процедуры)
rebuild procedure u1.cnt;
! Должно быть  NULL -  таблица изменена
execute u1.cnt();
username U1/12345678
rebuild procedure u1.cnt;
! Должно быть 3
execute cnt();
username U2/12345678
! Должно быть 3
execute u1.cnt();
username SYSTEM/MANAGER8
drop user u1 cascade;
drop user u2 cascade;
```

Определение привилегий на хранимую процедуру

Функция

Определение оператора задания привилегии на выполнение хранимой процедуры.

Спецификация

[1] <определение привилегий на хранимую процедуру>::=
GRANT <привилегия> ON [<имя схемы>.]<имя хранимой процедуры>
TO {PUBLIC | {<имя пользователя> | <имя роли>}[, …]}
[2] <привилегия>::= {EXECUTE | EXECUTE AS OWNER}

Общие правила

1) Опция `PUBLIC` предоставляет доступ к процедуре всем пользователям БД.
2) Опции `<имя пользователя>` и `<имя роли>` предоставляют доступ к процедуре соответственно указанному пользователю БД и пользователям, которым назначена роль.
3) Давать права на выполнение хранимой процедуры может только ее владелец.
4) Привилегия `EXECUTE` дает право на выполнение процедуры от своего имени (то есть выполнять от своего имени все содержащиеся в ней SQL-запросы, как это делает предложение `EXECUTE <имя хранимой процедуры>`).
5) Привилегия `EXECUTE AS OWNER` дает право выполнять процедуру от имени ее владельца (то есть все содержащиеся в процедуре SQL-запросы будут выполняться от имени владельца процедуры).

Пример

```
username SYSTEM/MANAGER8
create or replace table AAA(i int);
create or replace procedure AAA_proc(in i int) result int
code
  print("Name of user: " + username()); //
  execute direct "insert into AAA(i) values (" + itoa(i) +
 ");"; //
  return i; //
end;
drop user "TEST" cascade;
create user "TEST" identified by '12345678';
grant execute as owner on SYSTEM.AAA_proc to "TEST";
grant select on AAA to "TEST";
username TEST/12345678
execute SYSTEM.AAA_proc(100) as owner;
select * from SYSTEM.AAA;
```

Отмена привилегий на хранимую процедуру

Функция

Определение оператора отмены привилегии на выполнение хранимой процедуры.

Спецификация

[1] <отмена привилегий на хранимую процедуру>::=
REVOKE <привилегия> ON [<имя схемы>.]<имя хранимой процедуры>
FROM {PUBLIC | {<имя пользователя> | <имя роли>}[, …]}
[2] <привилегия>::= {EXECUTE | EXECUTE AS OWNER}

Общие правила

1) Опция `PUBLIC` запрещает доступ к процедуре всем пользователям БД, кроме ее владельца.
2) Опции `<имя пользователя>` и `<имя роли>` закрывают доступ к процедуре соответственно указанному пользователю БД и пользователям, которым назначена роль.
3) Отнимать права на выполнение хранимой процедуры может только ее владелец.
4) Опция `EXECUTE` отнимает права на выполнение процедуры от своего имени (то есть выполнять от своего имени все содержащиеся в ней SQL-запросы, как это делает предложение `EXECUTE <имя хранимой процедуры>`).
5) Опция `EXECUTE AS OWNER` отнимает права на выполнение процедуры от имени ее владельца (то есть выполнять все содержащиеся в процедуре SQL-запросы от имени владельца процедуры).

Выполнение хранимой процедуры

Определение оператора выполнения хранимой процедуры.

Спецификация

[1] <выполнение хранимой процедуры>::=
{EXECUTE | CALL} [<имя схемы>.]<имя хранимой процедуры>
[([<параметр> [, …]])] [AS OWNER] [INTO <имя переменной>[, …]]

Синтаксические правила

1) Конструкция `<выполнение хранимой процедуры>` исполняет предварительно оттранслированную и хранящуюся в БД процедуру. Любые виды рекурсивных вызовов разрешаются.
2) `<Список параметров>` – это список выражений, имен переменных или SQL-параметров, разделенных запятыми (может быть и пустым). Фактические параметры ставятся в соответствие формальным по порядку следования.

Примечание
SQL-параметр нельзя устанавливать в соответствие параметру хранимой процедуры с типом данных `CURSOR`.

3) Количество параметров процедуры в вызове не должно превышать количество параметров в описании процедуры.
4) 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.
5) Если формальные параметры вызываемой процедуры имеют модификатор `IN`, то в качестве фактических параметров можно использовать любые выражения

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

(совместимость типов выражений с типами параметров проверяется на этапе выполнения процедуры).

6) 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.

7) При указании опции AS OWNER и наличии прав на выполнение процедуры с правами владельца, процедура будет исполняться с правами владельца, все выполняемые запросы и вызовы процедур внутри процедуры при отсутствии указания схемы будут выполняться в схеме владельца. Для получения имени и идентификатора пользователя, запустившего её на выполнение, необходимо использовать функции username() и userid() соответственно, а для получения имени и идентификатора пользователя, от имени которого выполняется процедура, необходимо использовать функции effective_username() и effective_userid() соответственно.

8) 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. пункт Открытие курсора).

9) 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, количество параметров и тип возвращаемого значения. В случае ошибок происходят исключения BADPARAM или BADRETVAL.

10) Оператор используется для вызова хранимой процедуры внутри другого объекта процедурного языка (хранимой процедуры, триггера или временной процедуры).

11) <Параметр> представляет собой совокупность литералов и/или значений параметров и/или произвольных выражений, разделенных запятой.

```
create or replace procedure test (in i int) result int
code
  return i;
end;
; Допустимая конструкция
execute test(1+2);
```

12) При наличии у параметров опции DEFAULT, они могут опускаться, в этом случае поле ввода параметра необходимо оставить пустым и ввод следующего параметра выполнить после запятой.

```
create procedure test_proc (in k int, in m int DEFAULT 2, in n
 int) …
call test_proc(1,,3);
```

13) Для передачи логических значений параметров используются целые числа (0 интерпретируется как FALSE, 1 – как TRUE) или символьные литералы 'true' и 'false' (в любом регистре).

14) В рамках одной транзакции первая запущенная процедура для выполнения оператора EXECUTE открывает от основного канала свой дочерний канал. Все последующие процедуры, вызываемые в рамках той же транзакции, переиспользуют тот же самый канал, не открывая новых. Поэтому COMMIT/ROLLBACK в процедуре влияет не только на изменения, сделанные данной процедурой, но и на все изменения, сделанные всеми вызванными ранее процедурами. Чтобы избежать

этого, процедура должна использовать точку сохранения (SAVEPOINT), установив ее в начале транзакции, и подавать команды COMMIT/ROLLBACK до нее.



Примечание

Поскольку триггеры работают точно так же, как процедуры, все, сказанное для процедур, верно и для триггеров. Кроме того, поскольку все изменения производятся триггерами по одному общему каналу, то в случае нарушения логики работы, обнаруженного триггером, он может подать ROLLBACK, откатывающий изменения всех вызванных ранее триггеров и обеспечивающий целостность в рамках запроса.

15) Для выполнения чужой процедуры необходимо явно указывать <имя схемы> и иметь соответствующее право на вызов процедуры, назначенное владельцем процедуры.
16) Если опция AS OWNER не задана, то вызывающему пользователю должна быть назначена привилегия EXECUTE на данную процедуру и обращение к таблицам внутри тела процедуры будет выполняться от имени вызывающего пользователя и соответственно у вызывающего пользователя должны быть назначены соответствующие привилегии на таблицы.
17) Если опция AS OWNER задана, то вызывающему пользователю должна быть назначена привилегия EXECUTE AS OWNER на данную процедуру и обращение к таблицам будет выполняться от имени владельца процедуры и права доступа к таблицам, используемым в процедуре, не требуются.

Примеры оператора выполнения процедуры

```
1)
call myproc("auto",,1,aa) into bb;
2)
create or replace procedure tst_param (in id int; in ch char(10);
 out answ
  char(20)) result int for debug
code
  answ:= tochar(id) + "  " + ch; //
  return 0; //
end;

call tst_param(?,:arg2);
235
abcd

output parameters ( 235  abcd )
3)
create or replace table T1 (id int primary key, s int);
insert into T1 values (2, 300);
insert into T1 values (7, 200);
insert into T1 values (4, 600);

create or replace procedure procedure1() result int for debug
declare
  var a int;//
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
code
  execute "select max(s) from SYSTEM.T1;" into a;//
  return a;//
end;

create or replace procedure procedure0() result int for debug
declare
  var a int;//
code
  call SYSTEM.procedure1() as owner into a;//
  return a;//
exceptions
  when all then resignal;//
end;

create or replace user U1 identified by '12345678';
grant resource to U1;
grant execute as owner on procedure1 to U1;
grant execute as owner on procedure0 to U1;

username U1/12345678

execute SYSTEM.procedure0() as owner;
|600|
```

Удаление текста хранимой процедуры

Определение оператора удаления исходного текста хранимой процедуры.

Спецификация

[1] <удаление текста хранимой процедуры>::=
ALTER PROCEDURE [<имя схемы>.]<имя хранимой процедуры> DROP SOURCE TEXT

Общие правила

1) Команда очищает BLOB-значение, хранящее исходный текст хранимой процедуры (оттранслированный код процедуры не очищается), в результате исходный текст процедуры нельзя просматривать, редактировать, отлаживать (с помощью отладчика хранимых процедур) и перетранслировать. При этом доступны остаются операции: выполнение, удаление, модификация с другим исходным текстом (`ALTER PROCEDURE`).

Удаление хранимой процедуры

Определение оператора удаления хранимой процедуры.

Спецификация

[1] <удаление хранимой процедуры>::=
DROP PROCEDURE [<имя схемы>.]<имя хранимой процедуры>

Общие правила

1) Право удалить хранимую процедуру имеет только ее владелец.
2) Администратор БД имеет возможность удалить одновременно все объекты (в том числе и хранимые процедуры) некоторого пользователя с помощью команды `DROP USER <имя пользователя> CASCADE`.

© Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Временные процедуры

В отличие от хранимых процедур, временные процедуры компилируются и выполняются за один шаг, поэтому для их работы отпадает необходимость создания объекта БД «процедура» и наличие в БД системных таблиц `$$$PROC` и `$$$PRCD`. Скомпилированная временная процедура помещается в рабочую область ядра СУБД ЛИНТЕР, откуда и извлекаются результаты её выполнения.

Выполнение временной процедуры

Определение оператора выполнения временной процедуры.

Спецификация

[1] <выполнение временной процедуры>::=
EXECUTE BLOCK [AUTHID {CURRENT_USER | DEFINER}]
[RESULT <тип данных>] [FOR DEBUG]
<процедурный блок>END

Синтаксические правила

Правила оформления `<процедурного блока>` приведены в разделе Общий вид процедурного блока.

Примечание
Опции `AUTHID` и `FOR DEBUG` не запрещены по синтаксису, но реально их использовать затруднительно.

Примеры

```
1) Временная процедура:
execute block result int
declare
  var i int; //
code
  i := 1; //
  print("execute block");//
  return i; //
end;

2) Временная процедура с SQL-запросом:
create or replace table test(i int);
execute block result int
declare
  var i int; //
code
  i := 12; //
  execute direct "insert into test values(1)";//
  execute "insert into test values(2)";//
  return i; //
end;
```

```
select * from test;

3) Временная процедура с курсором:
execute block result cursor(i int)
declare //
  var b typeof(result); //
code //
  open b for "select i from test;"; //
  return b; //
end;

4) Рекурсивный вызов временных процедур:
execute block result int
declare
  var i int; //
code
  i := 123; //
  execute direct "execute block result int for debug declare var i
 int; //
                  code
                    i:=12; //
                    execute direct \"insert into test
 values(3)\"; //
                    execute \"insert into test values(4)\"; //
                    return i; //
                  end;"; //
  return i; //
end;
select * from test;
```


© Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Предопределенные переменные триггера

Старое и новое значение поля в триггере

Внутри тела триггера доступны 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 <статус поля>.<имя поля>. <Имя поля> соответствует имени поля таблицы. <Статус поля> – идентифицирует новое или старое значение этого поля (указано во фразе OLD [AS] и NEW [AS] предложения CREATE TRIGGER, или OLD и NEW 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.

Для полей типа BLOB допускается работа с OLD и NEW значениями не более 4000 байт, при превышении указанного размера будет происходить усечение значения. В триггере с опцией AFTER UPDATE значение OLD поля BLOB-типа недоступно.

Пример

```
create or replace table journal(table_name char(66), operation
 char(20),row_id int,time date,comment char(100));
create or replace trigger check_auto before update on auto for
 each row execute
code
  if old.personid <> new.personid then
    execute "select personid from person where personid= ?;" using
 new.personid; //
    if errcode() = 2 then
      execute "insert into journal values
 ('AUTO','UPDATE', ?,sysdate,'update to bad personid - IGNORED');"
 using old.personid; //
      return false; //
    endif; //
  endif; //
end;
update auto set personid = 1001 where personid = 2;
select * from journal;
|AUTO|UPDATE|2|03.11.2017:16:57:35|update to bad personid -
 IGNORED|
```

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, DELETING, UPDATING.

Триггерные предикаты – это предопределенные переменные, которые, в зависимости от события, принимают одно из значений TRUE или FALSE (таблица 3). С их помощью можно осуществлять различную обработку для различных событий. Например, вместо трех индивидуальных триггеров на разные события можно написать один триггер, который, в зависимости от события, будет выполнять ту или иную операцию.

Таблица 3. Принимаемые значения триггерных предикатов

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
code
  if (inserting) then
    execute "insert into result values ('inserting');"; //
  endif; //
  if (updating) then
    execute "insert into result values ('updating');"; //
  endif; //
  if (deleting) then
    execute "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';
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
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 записей 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 execute FOR DEBUG
code
  execute "insert into test_result(ch) values(?);" using
 itoa(rowcount); //
  execute "select make from auto;"; //
  execute "insert into test_result(ch) values(?);" using
 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 процедурного блока

Определение описания процедурного блока.

Спецификация

[1] <процедурный блок>::=
[<блок описаний>]
<блок кода>
[<блок обработки исключений>]
END;

Формат блока описаний

Определение блока описаний процедурного блока.

Спецификация

[1] <блок описаний>::=
DECLARE [<описание переменной>|<описание исключения>][...]
[2] <описание переменной>::=
VAR <имя переменной> {<простой тип данных>
[DEFAULT <инициализатор>] | <составной тип данных>};
[3] <описание исключения>::=
EXCEPTION <имя исключения> FOR {<вид исключения>|<код завершения>};

Синтаксические правила

1) Описание переменных простого типа аналогично описанию группы параметров:

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

2) Обращение к элементам массива выполняется с помощью квадратных скобок. Например, `z[3]`.

3) Присвоение NULL-значения всему массиву (например, `z:=NULL`) полностью очищает массив (размерность массива становится равной 0).

4) Массивы безразмерные, при обращении к отсутствующим элементам возвращается NULL-значение.

5) Индексом массива являются неотрицательные значения типа `SMALLINT`.

6) Не поддерживается:

- описание массивов в массиве (синтаксис `array array`);
- описание массива внутри курсора (синтаксис `array cursor`);
- описание массива курсоров (синтаксис `array array cursor`).

7) При описании массива байт (`array byte (128)`) невозможно обратиться к отдельному байту (получается массив массивов, трудности с отсутствием иерархии в синтаксисе), то есть синтаксис `z[3][5]` недопустим.

Пример

```
declare var z array varchar(256);
```

8) Перечень видов исключений процедурного языка и их десятичных кодов приведены в таблице 4.

Таблица 4. Перечень <видов критичных исключений> процедурного языка и их десятичных кодов

Вид исключения	Код	Описание
DIVZERO	-2	Деление на ноль
UNDEFPROC	-3	Попытка вызова неопределенной хранимой процедуры или отсутствие привилегии на вызов хранимой процедуры
BADPARAM	-4	Неверный параметр хранимой процедуры или стандартной функции.
BADINDEX	-5	Неправильный индекс массива или строки байт (BYTE, VARBYTE)
BADRETVAL	-6	Недопустимый тип возвращаемого хранимой процедурой значения
NULLDATA	-7	Попытка выполнить недопустимую операцию с NULL-значением (вычисление выражения при отсутствии данных)
NOMEM	-8	Не удается выделить память под переменные (обычно – превышен общий размер памяти, размер отдельной переменной и т.п.)
BADCURSOR	-9	Несоответствие структуры курсора количеству и/или типам записей запроса (при открытии или возврате курсора)
CURNOTOPEN	-10	Попытка выполнить `FETCH` или `CLOSE` для курсора, который не был открыт
BADCODE	-11	Неверный байт-код скомпилированной хранимой процедуры (устраняется перекомпиляцией процедуры)
TRIGQUERY	-12	Внутренняя ошибка СУБД: Вызов SQL-запроса в хранимой процедуре, которая в данном контексте должна вернуть значение, не обращаясь к БД
APPENDNOTSTARTED	-14	Перед подачей putm не было вызвано оператора **start append**, или он завершился неудачно
QUERYWHENAPPEND	-15	Между **start append** и **end append** нельзя использовать **execute**
APPENDACTIVE	-16	Попытка выполнить **start append**, когда предыдущий **start append** не был завершен при помощи **end append**
APPLICATIONERROR	-17	Сообщение сгенерировано функцией **raise_error()**
INVTRSTATE	-18	Ошибка с состоянием транзакций (чаще всего – количество открытий не совпадает с количеством закрытий транзакции)
OVERFLOW	-19	Переполнение типа переменной
CUSTOM <число>	-100	Код завершения пользовательского исключения СУБД ЛИНТЕР

© Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

9) <Код завершения> – коды завершения СУБД ЛИНТЕР (см. документ «Справочник кодов завершения», раздел «Коды завершения компилятора процедурного языка (10000-10086)»).
10) Максимальная длина строковой переменной процедурного языка 3919 символов. Но если в операторе

```
execute direct "insert into table_results values('"+str+"');";
```

строковой переменной str присвоить значение длиной 3900 символов (меньше, чем максимальная длина), то всё равно возникнет исключение NOMEM, потому что значение, которое стоит после execute direct – это тоже строковое значение, и получаемая в результате суммарная длина строки "insert into table_results values" и переменной str превышает допустимое значение.
11) 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):

```
CUSTOM <код>
```

12) Пользовательские исключения никогда не совпадают с кодами завершения СУБД ЛИНТЕР (они хранятся со знаком минус).
13) Необработанное исключение в триггере эквивалентно FALSE (запрету). После возникновения исключения в одном из триггеров (если они могут запретить операцию) выполнение других триггеров продолжается. То есть все триггеры будут выполнены (однако, если один из BEFORE-триггеров запретил операцию, то операция не выполнится, и соответствующие AFTER-триггеры тоже не выполнятся).

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

Определение описания блока кода процедурного языка.

[1] <блок кода>::=
CODE [<операторы>] | [<вложенный блок>] [<блок обработки исключений>] END;
[2] <вложенный блок>::=
CODE [<операторы>] | [<вложенный блок>] [<блок обработки исключений>] END[;]

Синтаксические правила

1) Представления тела процедуры в виде иерархии вложенных блоков возникает при необходимости локализации обработки исключений, например, исключение NULLDATA может генерироваться в каждом вложенном блоке и в каждом из них должно обрабатываться по собственному правилу.
2) Если вложенный блок содержит <блок обработки исключений>, или блок обработки содержит оператор RESIGNAL, то возникшие в этом вложенном блоке исключения обрабатываются сначала в его <блоке обработки исключений>.

Если вложенный блок не содержит <блок обработки исключений> или исключение не может быть обработано в его <блоке обработки исключений>, то при запуске ядра СУБД ЛИНТЕР с ключом /COMPATIBILITY=RESIGNAL_ALL исключение передается по иерархии на следующие вложенные блоки (то есть при таком запуске ядра СУБД нет необходимости в каждом вложенном блоке явно выполнять оператор RESIGNAL), в противном случае обработка исключения сразу передается на <блок обработки исключений> самого верхнего уровня (если этот <блок обработки исключений> задан).

3) На любом уровне вложенности при отсутствии ключа /COMPATIBILITY=RESIGNAL_ALL все неопределенные в блоке описаний исключения, кроме критичных для исполнения, игнорируются. Критичными считаются следующие исключения: DIVZERO, UNDEFPROC, BADPARAM, BADRETVAL, BADCURSOR, CURNOTOPEN, NOMEM, OVERFLOW. Если происходит такое исключение, и для него нет обработчика, автоматически выполняется завершение процедуры с передачей исключений выше, в блок вызвавшей процедуры, из которого произошел вызов (с NULL-значением в качестве возвращаемого значения) (см. также оператор RESIGNAL).
4) Исключения, не относящиеся к списку критичных (DIVZERO, UNDEFPROC, BADPARAM, BADRETVAL, BADCURSOR, CURNOTOPEN, OVERFLOW), например, NULLDATA, при отсутствии ключа /COMPATIBILITY=RESIGNAL_ALL, возникшие в блоке CODE...END, не попадут в обработчик предыдущего уровня.

Например, процедура

```
CREATE OR REPLACE PROCEDURE TST_CODE() RESULT INT FOR DEBUG
DECLARE
  VAR v, z int;
  EXCEPTION NULLDATA for NULLDATA;
CODE
  v:=3;
  CODE
    z:= NULL;
    v:= z + v;
  END
  RETURN 0;
EXCEPTIONS
  WHEN NULLDATA THEN
    RETURN 1;
END;

CALL tst_code();
result=1
```

вернёт результат 1, так как во вложенном блоке CODE... END нет своего обработчика исключений, и сгенерированное в нем исключение будет передано на обработку на предшествующий по иерархии уровень.

А процедура:

```
CREATE OR REPLACE PROCEDURE tst_code() RESULT INT FOR DEBUG
DECLARE
  VAR v, z INT;
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
  EXCEPTION NULLDATA FOR NULLDATA;
CODE
  v:=3;
  CODE
    z:= NULL;
    v:= z + v;
  EXCEPTIONS
    WHEN NULLDATA THEN
      RETURN 0;
  END
EXCEPTIONS
  WHEN NULLDATA THEN
    RETURN 1;
END;

CALL tst_code();
result=0
```

вернёт результат 0, так как во вложенном блоке `CODE… END` есть свой обработчика исключений, поэтому сгенерированное в нем исключение не будет передано на обработку на предшествующий по иерархии уровень.

Примеры

```
1)
create or replace procedure tst_code() result int for debug
declare
  var make char(20);
  exception no_data_found for 2;
  exception notab for 2202;
  var val_ret int;
-- code1 - первый вложенный блок
code
  code
    execute "select distinct model from auto where make like 'F
%'and personid=90;" into make;
  exceptions
    when no_data_found then
      val_ret:=-1;
      goto ret;
    when notab then
     val_ret:=-2;
     goto ret;
  end
-- code2 - второй вложенный блок
code
  execute "select distinct model from auto where make like 'F%'and
 personid=-1" into make;
```

```
  exceptions
    when no_data_found then
      val_ret:=-3;
      goto ret;
    when notab then
      val_ret:=-4;
      goto ret;
  end
  ret: return val_ret;
end;

call tst_code();
result -3

2)
create or replace procedure tst_code() result int for debug
declare
  var make char(20);
  exception no_data_found for 2;
  exception notab for 2202;
  var val_ret int;
// code1 - первый вложенный блок
code
  val_ret:=100;
  code
    execute "select distinct model from auto where make like 'F
%'and personid=-1;" into make;
    code
      execute "select distinct model from auto where make like 'F
%'and personid=90;" into make;
    exceptions
      when no_data_found then
        val_ret:=-3;
        goto ret;
      when notab then
        val_ret:=-4;
        goto ret;
    end
  exceptions
    when no_data_found then
      val_ret:=-1;
      goto ret;
    when notab then
      val_ret:=-2;
      goto ret;
  end
```

```
  ret: return val_ret;
end;

call tst_code();
result -1
```

Формат блока обработки исключений

Определение описания блока обработки исключений.

[1] <блок обработки исключений>::=
EXCEPTIONS
WHEN <имя исключения>[, …] THEN
[<операторы> | IGNORE]
[WHEN <имя исключения>[, …] THEN
[<операторы> | IGNORE]]
… [WHEN OTHERS THEN
[<операторы> | IGNORE]]
[WHEN ALL THEN
[<операторы> | IGNORE]]

Синтаксические правила

1) <Имя исключения>[, …] – это одно или более определенных в блоке описаний имен, разделенных запятыми.
2) При возникновении исключения переменной, при обработке которой возникло это исключение, присваивается NULL-значение.
3) 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.
4) Фраза WHEN OTHERS THEN <операторы> задает обработку всех декларированных в блоке описаний исключений, обработка которых не предусмотрена в предыдущих фразах WHEN....
5) 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).
6) Оператор IGNORE заставляет процедуру игнорировать перечисленные критичные исключения. Критичными считаются исключения: DIVZERO, UNDEFPROC, BADPARAM, BADRETVAL, BADCURSOR, CURNOTOPEN, OVERFLOW. 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). Выполнение процедуры может быть продолжено с помощью оператора GOTO.
7) Если конструкции WHEN, 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,

OVERFLOW. 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…

```
create or replace 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
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
    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;
```

Глобальные переменные хранимых процедур

Процедурный язык СУБД ЛИНТЕР поддерживает механизм глобальных переменных процедурного языка (см. документ «Архитектура СУБД», пункт «Глобальные переменные хранимых процедур»).

В процессе выполнения хранимых процедур значения глобальных переменных хранятся в локальной для каждой пользовательской сессии области глобальных переменных, то есть одна и та же глобальная переменная в каждой пользовательской сессии будет иметь свою копию. Это означает, что значение одной и той же глобальной переменной индивидуальны в каждой пользовательской сессии и не доступны в других пользовательских сессиях. Для доступа к общим данным из разных пользовательских сессий необходимо использовать специальные пользовательские таблицы БД.

Для эффективного доступа к глобальным переменным в области глобальных переменных ведется хеш-таблица переменных, элементами которой является смещение начала цепочки переменных с данной хеш-суммой в области переменных. Хеш-сумма переменной вычисляется путем побайтной операции XOR идентификатора переменной (а не имени переменной).

Специальный оператор декларирования глобальных переменных в процедурном языке СУБД ЛИНТЕР не используется – глобальной переменной объявляется любое неопределенное имя (среди локальных переменных, функций и т.д.). Просмотр таких неопределенных имен выполняется с помощью SQL-запроса к системной таблице глобальных переменных и при удачном поиске найденная переменная включается в список переменных хранимой процедуры с значением по умолчанию (если оно было задано при создании/модификации глобальной переменной). Если значение по умолчанию отсутствует, то переменная перед использованием должна быть инициализирована.

Изменения глобальных переменных, производимые в одной пользовательской сессии, не видны в других пользовательских сессиях – в каждой пользовательской сессии используется свой локальный набор текущих значений одних и тех же глобальных переменных, поэтому для обмена информацией между одновременно работающими пользователями следует использовать не глобальные переменные, а специальные пользовательские таблицы.

Глобальные переменные нельзя использовать в качестве параметров любого типа хранимых процедур.

Создание/модификация/удаление глобальных переменных выполняется с помощью соответствующих SQL-конструкций (см. документ «Справочник по SQL», раздел «Глобальные переменные процедурного языка»).

Примеры

```
1)
create if not exists variable abc int = 10;

create or replace procedure rabc() result int
code
  return abc;
end;
```

```
execute rabc();

return value = 10

2)
create or replace variable Org varchar(256) ='АО НПП «РЕЛЭКС»';

create or replace procedure company() result varchar(20)
code
  return Org;
end;
execute company();

return value = АО НПП «РЕЛЭКС»

3)
create or replace variable invoice varchar(20)='A/5211';
create or replace variable date_invoice date=to_date('23.04.2014',
 'dd.mm.yyyy');

create or replace procedure pay() result varchar(50)
declare
  var ret varchar(50); //
code
  ret := "Номер счета: " + invoice +". Дата: "+ dtoa(date_invoice,
 "dd.mm.yyyy"); //
  return ret;  //
end;
execute pay();
return value = Номер счета: A/5211. Дата: 23.04.2014

4)
create variable abcd int = 1;
create or replace procedure ZZZ_1() result int
code
  return abcd;//
end;

select count(*) from $$$PRCD where PROCID
  IN (select $$$ID from $$$PROC where $$$NAME LIKE '%ZZZ_1%');

|          1|

5)
create or replace variable abcd int = 1;
create or replace procedure ZZZ_1() result int
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
code
  abcd:=abcd+1000; //
  return abcd;//
end;

execute ZZZ_1();
return value = 1001
```

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.

Синтаксис

```
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.

Синтаксис

```
CASE <выражение>
  WHEN <литерал1 [, литерал]... [, литерал]> THEN
    <операторы 1>
  WHEN <литерал2 [, литерал]... [, литерал]> THEN
    <операторы 2>
.............
  [WHEN OTHERS THEN
    <операторы>
  ]
ENDCASE
```

Описание

Тип литералов и тип CASE-выражения должны быть совместимы.

Значение CASE-выражения последовательно сравнивается с литералами в операторах WHEN. При первом совпадении значения выражения CASE с литералом одного из оператора WHEN выполняются операторы, соответствующие этому условию (то есть до следующего оператора WHEN или до ENDCASE), после чего выполнение оператора выбора завершается. Если значение выражения CASE не было найдено среди списка литералов, 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;
```

Операторы цикла

Оператор цикла с предусловием

Назначение

Оператор цикла с предусловием предназначен для организации многократного выполнения фрагмента программы при выполнении некоторого условия, указанного перед его началом. Это условие проверяется до выполнения <тела цикла>, поэтому <тело цикла> может быть не выполнено ни разу (если условие с самого начала ложно).

Синтаксис

```
WHILE <выражение> LOOP
  <тело цикла>
ENDLOOP
```

Описание

<выражение> должно быть логического типа. Если его значение `TRUE`, выполняется <тело цикла> до `ENDLOOP`, после чего снова вычисляется <выражение> и принимается решение о продолжении (прекращении) цикла. Если значение <выражения> `FALSE`, управление передается на следующий после `ENDLOOP` оператор. Чтобы количество циклов было конечным, в <теле цикла> обязательно должно выполняться изменение переменных или условий, влияющих на <выражение>.

Пример

Классический перебор записей в выборке.

```
create or replace table tab1 (i int, j int, s int);
insert into tab1 (i,j) values (1, 50);
insert into tab1 (i,j) values (2, 30);
insert into tab1 (i,j) values (3, 60);
insert into tab1 (i,j) values (4, 40);

create or replace procedure proc1 () result int for debug
declare
  var sum int;//
  var CURS cursor (i int, j int, s int);//
code
  sum := 0;//
  open curs as "curs" for "select * from tab1;";//
  fetch curs first;//
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
  while not outofcursor(curs) loop
    sum := sum + curs.i + curs.j;//
    execute "update tab1 set s = ? where current of \"curs\";",
 itoa(sum);//
    fetch curs next;//
  endloop
  return sum;//
exceptions
  when all then resignal;//
end;
--
call proc1();
Результат 190

--
select * from tab1;
I   J   S
1  50  51
2  30  83
3  60  146
4  40  190
```

Оператор цикла с постусловием

Назначение

Оператор цикла с постусловием предназначен для организации многократного выполнения фрагмента программы при выполнении некоторого условия, указанного после выполнения <тела цикла>. Это условие проверяется после выполнения <тела цикла>, поэтому <тело цикла> всегда будет выполнено хотя-бы один раз.

Синтаксис

```
LOOP
  <тело цикла>
UNTIL <выражение>
```

Описание

<выражение> должно быть логического типа. Если его значение `TRUE`, выполняется <тело цикла> начиная с `LOOP`, после чего снова вычисляется <выражение> и принимается решение о продолжении (прекращении) цикла. Если значение <выражения> `FALSE`, управление передается на следующий после `UNTIL` оператор. Чтобы количество циклов было конечным, в <теле цикла> обязательно должно выполняться изменение переменных или условий, влияющих на <выражение>.

Пример

```
x:=1;
loop
  {тело цикла}
  x:=x+1;
```

```
until x>20;
```

Оператор цикла с параметром

Синтаксис

```
FOR <инициализатор1>{,<инициализатор2>}
WHILE <выражение> [BY <инкремент1>{,<инкремент2>}] LOOP
   <тело цикла>
ENDLOOP
```

Описание

<инициализатор1> и <инициализатор2> – операторы присвоения начальных значений параметрам цикла.

<инкремент1> и <инкремент2> – операторы изменения значений <инициализатора1> и <инициализатора2> соответственно.

<выражение> должно быть логического типа.

До начала цикла выполняется присвоение значений <инициализатору1> и, если задан, <инициализатору2>, после чего проверяется значение <выражение>. Если его значение TRUE, выполняется <тело цикла>, начиная с LOOP. После чего выполняется изменение параметров цикла (если задана конструкция BY ...) заново вычисляется <выражение> и принимается решение о продолжении (прекращении) цикла. Если значение <выражения> FALSE, управление передается на следующий после ENDLOOP оператор.

Чтобы количество циклов было конечным, в <теле цикла> или в конструкции BY ... обязательно должно выполняться изменение значений параметров или условий, влияющих на <выражение>.

Примеры

```
1)
for i:=1 while i<=10 by i:=i+1 loop
  x[i]:=0;
endloop

Данная конструкция аналогична следующим:
for i:=1 while i<=10 loop
  x[i]:=0;
  i:=i+1;
endloop;

i:=1;
while i<=10 loop
  x[i]:=0;
  i:=i+1;
endloop;

2)
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
for i:=0, n:=1 while i<8 by i:=i+1, n:=n*2 loop
endloop;
```

Досрочное завершение текущего цикла

Назначение

Досрочное завершение выполнения циклического оператора.

Синтаксис

```
BREAK;
```

Описание

Оператор заканчивает выполнение всего ближайшего внешнего цикла или условного оператора, в котором он задан. Управление передаётся оператору, который следует за завершающим оператором, если таковой имеется.

В случае вложенных циклов оператор немедленно прекращает выполнение самого внутреннего из объемлющих его циклов.

Досрочный переход к следующей итерации цикла

Назначение

Досрочное завершение выполнения текущей итерации циклического оператора.

Синтаксис

```
CONTINUE;
```

Описание

Оператор досрочно завершает выполнение текущей итерации цикла и переходит к следующей итерации того же цикла, в теле которого он задан (все операторы до конца тела цикла пропускаются, то есть он как бы имитирует безусловный переход на конечный оператор цикла, но не за пределы самого цикла).

Безусловный переход

Назначение

Оператор безусловного перехода предназначен для управления выполнением программы.

Синтаксис

```
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;
```

Возврат из процедуры

Синтаксис

```
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.

При отсутствии в процедуре оператора RETURN результат будет NULL, при отсутствии в триггере оператора RETURN результат будет TRUE.

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

Синтаксис

```
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

Синтаксис

```
RESIGNAL;
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Описание

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

Все возникшие в процедуре исключения (кроме нескольких критичных, которые указаны в таблице Виды критичных исключений) не передаются на верхний уровень, если явно не вызывается RESIGNAL или, если ядро СУБД ЛИНТЕР не запущено с ключом /COMPATIBILITY=RESIGNAL_ALL.

Открытие курсора

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

Синтаксис

```
<курсор с претранслируемым запросом> ::=
OPEN <имя курсорной переменной> [AS "имя курсора"] FOR <запрос>
 [{,|USING} <параметр> [, …]];

<курсор с динамическим запросом>::=
OPEN <имя курсорной переменной> [AS "имя курсора"] FOR DIRECT
 <выражение символьного типа> [{,|USING} <параметр> [, …]];

<курсор с процедурой, возвращающей курсор>::=
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.

Формат <запроса> рассмотрен в разделе Выполнение запроса. 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 первой записи выборки уже доступны через курсорную переменную.

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

Примеры

```
1) курсор с претранслируемым запросом:
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) курсор с динамическим запросом:
create or replace procedure tst(in id int) result cursor(c
 char(20))
declare
  var b typeof(result);
code
  open b for direct "select make from auto where personid=" +
 itoa(id) + ";";
  return b;
end;

3) курсор с процедурой, возвращающей курсор:
open b as "cursor_b" for call prc_retcur("auto");
```

Выборка данных из открытого курсора

Синтаксис

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

Описание

Данный оператор выбирает очередную запись из курсора согласно ориентации. Записи курсора могут содержать не более 255 столбцов. Результаты записи 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 значений:

- NEXT – следующая запись;
- PREVIOUS – предыдущая;
- FIRST – первая;
- LAST – последняя;

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

- ABSOLUTE <выражение> – запись с номером, вычисленным в выражении (типа INTEGER или типа SMALLINT);
- RELATIVE <выражение> – запись, расположенная на вычисленном в выражении расстоянии от текущей записи (расстояние должно быть целочисленным).

Если ориентация не задана, предполагается значение NEXT.

Для организации цикла по выборке используется комбинация операторов FETCH и обычного цикла WHILE, в условии которого стоит вызов стандартной функции outofcursor(). Кроме outofcursor есть еще функции rowcount() для получения количества записей выборки данных и errcode() для получения кода завершения (в случае, если он не перехватывается блоком обработки исключений).

```
create or replace procedure example_fetch() result int
declare
  var s cursor(nm char(20));//
code
  print ("Список моделей авто");//
  open s for "select distinct model from auto;";//
  while not outofcursor(s) loop
    print (s.nm);//
    fetch s;//
  endloop
  return 0;//
end;
Результат:
*** Message from Sored Procedure: Список моделей авто
*** Message from Sored Procedure: 124 SPORT COUPE
*** Message from Sored Procedure: 1275 GT
…
```

Описание конструкции INTO <переменная> [, …] приведено в пункте Выполнение запроса.

При попытке сделать FETCH для неоткрытого курсора генерируется исключение CURNOTOPEN.

Если при запуске ядра СУБД был задан ключ /COMPATIBILITY=NOREC_EXCEPTION (см. документы «Запуск и останов СУБД ЛИНТЕР в среде ОС Windows», раздел «Ключи совместимости с SQL-стандартом и другими СУБД»«Запуск и останов СУБД ЛИНТЕР в среде ОС Windows», раздел «Ключи совместимости с SQL-стандартом и другими СУБД» и «Запуск и останов СУБД ЛИНТЕР в среде ОС Linux», раздел «Ключи совместимости с SQL-стандартом и другими СУБД»«Запуск и останов СУБД ЛИНТЕР в среде ОС Linux», раздел «Ключи совместимости с SQL-стандартом и другими СУБД» «Запуск и останов СУБД ЛИНТЕР в среде ОС Linux», раздел «Ключи совместимости с SQL-стандартом и другими СУБД»), то при выполнении FETCH при отсутствии следующей записи в курсоре будет генерироваться исключение 2 (NO_DATA).

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.

Синтаксис

```
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 запроса

Назначение

Оператор запроса предназначен для организации непосредственной работы с СУБД.

В хранимых процедурах допустимы два вида запросов: претранслируемые, которые разбираются на этапе трансляции процедуры, и динамические, которые формируются, транслируются на этапе выполнения процедуры и сразу же выполняются. Приоритет обоих типов запросов наследуется от родительского канала. Претранслируемые и динамические запросы делятся на запросы возвращающие выборку данных и не возвращающие выборку данных. Запросы, возвращающие выборку данных, могут быть использованы в курсорах (см. пункт Открытие курсора). Текст запроса должен оканчиваться знаком (;) и быть заключенным в двойные апострофы (").

В качестве претранслируемых запросов обычно используются запросы с параметрами. Запрос транслируется и сохраняется в БД. На этапе выполнения к претранслированному запросу привязываются вычисленные параметры, если они есть.

В качестве динамических запросов используются запросы, текст которых на момент трансляции процедуры неизвестен (он формируется в виде символьного выражения хранимой процедурой в процессе её выполнения, поэтому синтаксический и семантический контроль такого запроса производится при выполнении процедуры).

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

Синтаксис

```
<выполнение претранслируемого запроса>::=
    EXECUTE <запрос> [{,|USING} <параметр> [, …]] [INTO
 <переменная> [, …]];
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
<выполнение динамического запроса>::=
    EXECUTE DIRECT <выражение символьного типа> [{,|USING}
 <параметр> [, …]] [INTO <переменная> [, …]];
```

Описание

1) 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).
2) <Запрос> должен быть обрамлен двойными апострофами «"» и оканчиваться «;».
3) 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 (при возвращении более одной записи в скалярную переменную будет сгенерировано исключение BADPARAM).

   Для <переменной> допускаются все простые типы данных, кроме BLOB и EXTFILE.

Примечание
Если результат <запроса> – пустая выборка, то <переменная> сохранит значение, которое было перед выполнением EXECUTE … INTO …

4) Если задана опция DIRECT, текст запроса при создании процедуры не проверяется, а передается напрямую ядру СУБД ЛИНТЕР при выполнении процедуры, то есть синтаксический и семантический контроль запроса выполняется при исполнении процедуры.

   Если опция DIRECT не указана, синтаксический и семантический контроль <запроса> выполняется на этапе создания процедуры.

   Сравните (таблица t1 отсутствует в БД):

```
CREATE OR REPLACE PROCEDURE tst_dir() RESULT NUMERIC for debug
DECLARE
  VAR VINST_ID NUMERIC;
CODE
  EXECUTE "SELECT a from t1;" INTO VINST_ID; //ошибка трансляции
  EXECUTE DIRECT "SELECT a from t1;" INTO VINST_ID; //без ошибки
 трансляции
  RETURN VINST_ID;
END;
```

5) Если заданы опции DIRECT и <параметры>, то во время выполнения оператора сначала выполняется трансляция <запроса>, затем привязка параметров и выполнение <запроса>.

Примечание
Рекомендуется использовать опцию DIRECT только в ситуациях, когда статическая компиляция запроса невозможна – например, при динамическом указании имени таблицы или имен столбцов. Нежелательно использование оператора EXECUTE с опцией DIRECT для подстановки параметров в запрос, поскольку это гораздо эффективнее выполняется при помощи статического EXECUTE.

6) В результате выполнения оператора EXECUTE может возникнуть исключение, соответствующее коду завершения СУБД ЛИНТЕР (так же, как и при выполнении операторов OPEN, FETCH и CLOSE).

7) Запрос может транслировать один пользователь, а выполнять другой (в случае, если запрос указан в команде EXECUTE, и процедура выполняется не в режиме AS OWNER). Проверки наличия привилегий при этом проверяются для выполняющего пользователя, а не для транслирующего. От транслирующего пользователя берутся только умолчания (прежде всего его текущая схема).

   Есть различие с действием команды EXECUTE DIRECT при выполнении процедуры не в режиме AS OWNER – там и права проверяются для выполняющего пользователя, и умолчания берутся его же.

```
Пользователь CREATOR: создает процедуру с командой:
EXECUTE "SELECT * FROM TBL;"
Пользователь EXECUTOR: выполняет эту процедуру без AS OWNER
Результат: запрос от имени EXECUTOR подается к таблице CREATOR.TBL

Пользователь CREATOR: создает процедуру с командой:
EXECUTE DIRECT "SELECT * FROM TBL;"
Пользователь EXECUTOR: выполняет эту процедуру без AS OWNER
Результат: запрос от имени EXECUTOR подается к таблице
 EXECUTOR.TBL
```

Примеры

```
1)
execute "update tab1 set s = ? where current of \"CURS\";" using
 sum;

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

3)
var cnt int; //
…
execute "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;
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
5)
create or replace table test(i int, utf nchar(10), v_utf
 nvarchar(20));
insert into test (i, utf, v_utf) values(1, n'342f', n'56ffca45');
insert into test (i, utf, v_utf) values(2, n'cccc',
 n'56745333fffa');

create or replace procedure prc_test() result nchar(20) for debug
declare
  var ch nvarchar(50); //
code
  execute "select v_utf from test where i=2;" into ch; //
  return ch; //
end;
execute prc_test();
Return value = 56745333fffa

6)
create table aaa (i int, ch char(3));
insert into aaa values(1, 'abc');
insert into aaa values(2, 'def');
insert into aaa values(3, 'ghi');
create or replace procedure prc_test(in n int) result char(15) for
 debug
declare
  var ch typeof(aaa.ch);//
code
  execute "select ch from aaa where i=:p1;" using n into ch;//
  return ch;//
end;
execute prc_test(2);
drop table aaa;

Результат работы примера:

Return value = def

7)
execute block result typeof( person.fmlystat )
declare
  var age integer;//
  var stat typeof(person.fmlystat);//
code
  execute "update person set age=40 where name='kim' and
 firstnam='eddie'";//
```

```
  execute "select age, fmlystat from person where name=? and
 firstnam=?" using "kim", "eddie" into age, stat;//
  execute "update person set age=? where name=? and firstnam=?"
 using age + 1, "kim", "eddie";//
  execute "select age from person where name=? and firstnam=?",
 "kim", "eddie" into age;//
  return stat;//
end;

8)
create or replace procedure report(in Модель char(15); in Цвет
 char(10))
result int for debug
declare
  var Количество int; //
  var Запрос char(100); //
code
  Запрос:="select count(*) from auto where model='"+ Модель+ "'and
 color='" + Цвет+"';";//
  execute direct Запрос into Количество; //
  return Количество; //
end;
execute report('PANTERA','BLACK');
return value = 5
```

Завершение транзакции

Назначение

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

Синтаксис операторов

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

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 может подать операторы `COMMIT` или `ROLLBACK` по своему каналу, что отразится на всех изменениях, сделанных как самой процедурой, так и всеми ее дочерними процедурами.

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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 после завершения транзакции. В противном случае он остается открытым, что позволяет быстро продолжить сеанс изменений.

Выражения

Выражение – это комбинация объектов процедурного языка, называемых *операндами* выражения. В качестве операндов могут выступать переменные, значения переданных параметров, отдельные поля структуры литерала, 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 литералы 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 литерала 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`).

Примечание
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 `NULL`.

Максимальная длина результата символьного или байтового выражения – 3910 символов (байтов) (для выражений типа `UNICODE` – 1955 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>
```


© Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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 будет `NULL`.

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.

Примечания

1. Максимальное число строк-операндов в операции конкатенации равно 255, максимальная длина результирующего значения не должна превышать максимальную длину типа CHAR (VARCHAR).
2. Если ядро СУБД запущено с ключом /COMPATIBILITY=ORACLE, то результатом конкатенации символьных значений с NULL-значением будет исходная символьная строка (а не NULL-значение).

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>
```

Поддерживаются 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.

Побитовые логические операции

Логическое «И»

Синтаксис

```
<число1> & <число2>
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

<число1> и <число2> – целочисленные числовые выражения.

Описание

Возвращается результат битовой логической операции «И» из двух значений <число1> и <число2>.

Если один из операндов <число1> или <число2> имеет NULL-значение, возвращается NULL-значение.

Пример

```
create or replace procedure SP_BITAND(in a int; in b int)
result int
code
  return (a & b); //
end;
execute SP_BITAND(0x0FF, 0xFF0);
return value = 240
```

Логическое «ИЛИ»

Синтаксис

```
 <число1> | <число2>
```

<число1> и <число2> – числовые выражения.

Описание

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

Если один из операндов <число1> или <число2> имеет NULL-значение, возвращается NULL-значение.

Пример

```
create or replace procedure SP_BITOR(in a int; in b int) result
 int
code
  return (a | b); //
end;
execute SP_BITOR(0x0FF, 0xFF0);
return value = 4095
```

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 наинизший 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.

Примечание

В связи с особенностью транслятора хранимых процедур выражения типа (d2 $ d1) - 1 или 1 - (d2 $ d1) не могут правильно обрабатываться. В таких случаях выражение необходимо вычислять с помощью нескольких операций.

© Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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.

Синтаксис

```
<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).

В качестве правой части присвоения может использоваться вызов процедуры:

```
p_var := proc(param1, param2);
```

альтернативным способом присвоения значения переменной при вызове процедуры является конструкция вида:

```
call proc(param1, param2) INTO p_var;
```

Примечание
В <блоке кода> триггера для присвоения значения переменной с использованием вызова процедуры допустимо использовать только конструкции вида:

```
call proc(param1, param2) INTO p_var;
```

Условные выражения

Синтаксис

```
EIF <s1><логическое выражение><s2> <выражение1>[ ELSE
 <выражение2>]
<s1>:=[
<s2>:=]
```

Описание

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.

Пример

```
create or replace procedure p_eif (
  in i_a int default 0; //
  in i_b int default 0) result int for debug
code
  return eif[ i_a < i_b ] i_a else i_b;  //
end;
call p_eif(3, 4);
// 3
call p_eif(3, 2);
// 2
```


© Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Пакетное добавление

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

В обычном режиме добавление строки выполняется с помощью SQL-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 столбцу "I"
  l char(20) column "lowercase" // будет соответствовать столбцу
 "lowercase"
  ins date column "INSERT" // будет соответствовать столбцу
 "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 завершения.

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

При попытке выполнить start append или execute, 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 оператора putm, попытавшегося сбросить заполненную внутреннюю страницу и столкнувшегося 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` или `start append`!

5) все операции `putm` выполняются по одному курсору, по тому же, что и `execute` в процедурах ("курсору по умолчанию").

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



Примечание
При пакетной вставке данных триггеры, настроенные на вставку данных, срабатывать не будут.

Пример

```
create or replace 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 "create table pm("
                "\"start\" int, si smallint, bi bigint,"
                "c char(20), vc varchar(30), "
                "\"THIS IS DATE\" date,"
                "r real, db double, dc numeric,"
                "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.

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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

Процедуры с курсорным параметром

Все типы входных параметров (IN, INOUT, OUT) могут быть объявлены с типом данных CURSOR.

Входной курсорный параметр должен быть предварительно открыт в процедуре, которая вызывает на выполнение процедуру с данным курсорным параметром.

Курсорный параметр не может быть SQL-параметром.

Входные курсоры (выборки данных) в качестве входных параметров имеет смысл использовать в следующих случаях:

1) SQL-запрос, формирующий выборку данных курсорного параметра, не может реализовать все необходимые условия для формирования полноценной выборки данных. В этом случае возвращаемая SQL-запросом выборка данных перед дальнейшей обработкой может подвергаться дополнительной фильтрации с помощью сложных специальных функций процедурного языка;
2) передаваемая выборка данных перед последующим использованием должна быть обработана (модифицирована, обогащена данными) в соответствии со встроенным алгоритмом обработки.

Примеры

Пусть есть таблица bank, содержащая данные об организациях и текущих денежных суммах на их расчетных счетах (баланс).

```
create or replace table bank(id_org int, summa numeric);
insert into bank (id_org, summa) values (1001, 27356.0), (2705,
 110227.15), (4903, 2.75);
select * from bank;
```

Есть процедура, которая выполняется в конце каждого календарного месяца и производит корректировку баланса организации с учетом начисленных этой организации налоговых вычетов, отчислений, возвратов и т.п. Вычисление суммы корректировки баланса является довольно сложной процедурой, но в данном примере эта процедура просто возвращает для организации якобы вычисленную сумму корректировки:

```
create or replace procedure nalog(in id_org int) result numeric
code
  if id_org=1001 then return(-1577.65); endif;
  if id_org=2705 then return(3000.5); endif;
  if id_org=4903 then return(-86664.35); endif;
end;
```

Алгоритм работы:

1) в процедуре tst_cursor:
   - открываем курсор для получения выборки данных обо всех организациях и их балансах;
   - вызываем процедуру upd_cursor, передавая ей в качестве входного параметра имя открытого курсора;

```
create or replace procedure tst_cursor() result int for debug
declare
  var curs cursor(id int, sm numeric);
code
  open curs for "select id_org, summa from bank;";
  call upd_cursor(curs);
  return 0;
end;
```

2) процедура upd_cursor:

- используя переданный ей курсорный параметр, организует цикл для перемещения по всем записям полученной выборки данных;
- для каждой организации (записи выборки данных) применяется процедура nalog для вычисления суммы отчислений/доначислений и производится корректировка баланса организации в курсорной переменной;
- новая сумма баланса организации записывается из курсорной переменной в таблицу bank.

```
create or replace procedure upd_cursor(in query cursor(ident int,
 current_sum numeric)) result int
declare
  var upd_summa numeric;
code
  while not outofcursor(query) loop
    upd_summa:= query.current_sum + nalog(query.ident);
    execute "update bank set summa = ? where id_org = ?;" using
 upd_summa, query.ident;
    fetch query;
  endloop
  return 0;
end;
```

Работа с типом данных BLOB

В процедурном языке работа с типом данных BLOB реализована с помощью нижеследующих функций (таблица 5).

Таблица 5. Функции для работа с типом данных BLOB

Имя	Функция
set_cur_blob	Установка текущего BLOB-столбца при обработке запроса с несколькими BLOB-значениями
add_blob	Добавление порции данных в конец BLOB-значения (запись в произвольное место не допустима)
read_blob	Чтение указанного типа данных из BLOB-значения
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	
seek_blob	Установка текущей позиции для чтения BLOB-данных
clear_blob	Удаление BLOB-значения
blob_size	Получение информации из описателя BLOB-значения
modify_blob_type	Типизация BLOB-значения
length	Длина BLOB-значения

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 определенной записи (`FETCH`);
- после добавления новой записи в таблицу.

Номер BLOB-столбца, к которому добавляется порция данных, предварительно устанавливается с помощью функции set_cur_blob. Если эта функция не была вызвана, то используется последний 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-данными существуют, 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-запроса 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 столбцу типа BLOB. При выборке записей SELECT-запроса для поля типа BLOB возвращается только информационный заголовок длиной 14 байт. Для работы собственно с BLOB-значениями (чтение, добавление, очистка) используются специальные стандартные функции. 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 BLOB-данных. 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 BLOB-значении, 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.

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 записей выборки данных на работу с заданным столбцом. Если значение параметра <номер> равно 0, то происходит переключение на первый столбец типа BLOB. Кроме того, после ее вызова сбрасывается внутренний счетчик позиции BLOB-значения для чтения, то есть функции типа read_blob начнут читать порцию данных с начала BLOB-значения (для изменения позиции чтения необходимо использовать функцию seek_blob).

Добавление данных в конец BLOB-значения

Синтаксис

```
add_blob([<курсор>,] <значение>[, <размер>][, <номер BLOB-
столбца>]);
```

<курсор> – курсорная переменная;

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

<размер> – целочисленное положительное выражение;

<номер BLOB-столбца> – порядковый номер BLOB-столбца в текущей записи выборки данных, отсчёт начинается с 1.

Описание

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

Параметр <значение> определяет добавляемую порцию данных.

Возможны два случая:

1) если это переменная типа BYTE, то 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, то к BLOB-значению добавляется двоичное представление порции данных 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 и VARCHAR записываются как два байта длины строки, два байта кодировки и соответствующее количество байт самой строки после них. В этом случае длину порции данных указывать не требуется, так как она известна. Для значений типа NCHAR и NVARCHAR запись аналогична, за исключением двух байт кодировки. Они отсутствуют, так как для данных типов она по умолчанию UTF-16.

Примечания

1. 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.
2. Все данные, добавленные в BLOB-значение, могут быть адекватно считаны с помощью функций чтения read_blob.

Параметр <размер> можно определить только для типа BYTE.

Для всех остальных типов поле <размер> интерпретируется как <номер BLOB-столбца>.

Возвращаемое значение

Функция возвращает код завершения СУБД ЛИНТЕР.

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Чтение данных из BLOB-значения на общем уровне

Синтаксис

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

<курсор> – курсорная переменная;

<буфер> – переменная типа BYTE/VARBYTE;

<размер> – целочисленное положительное 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-значения считывается порция данных, меньшая или равная по длине значению параметра <размер> (размер порции меньше этого параметра, если в BLOB-значении 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 данных типа int из BLOB-значения

Синтаксис

```
read_blob_int([<курсор>]);
```

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из BLOB-значения заданного BLOB-столбца текущей записи данных типа INT.

Предполагается, что в BLOB-значении в текущей позиции для чтения находится двоичное представление данных типа INT, как описано в функции add_blob.

Возвращаемое значение

1) Считанные данные типа INT.
2) Код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Чтение данных типа smallint из BLOB-значения

Синтаксис

```
read_blob_smallint([<курсор>]);
```

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из BLOB-значения заданного BLOB-столбца текущей записи данных типа smallint.

Предполагается, что в BLOB-значении в текущей позиции для чтения находится двоичное представление данных типа smallint, как описано в функции add_blob.

Возвращаемое значение

1) Считанные данные типа SMALLINT.
2) Код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Чтение данных типа bigint из BLOB-значения

Синтаксис

```
read_blob_bigint([<курсор>]);
```

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из BLOB-значения заданного BLOB-столбца текущей записи данных типа bigint.

Предполагается, что в BLOB-значении в текущей позиции для чтения находится двоичное представление данных типа bigint, как описано в функции add_blob.

Возвращаемое значение

1) Считанные данные типа BIGINT.
2) Код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Чтение данных типа real из BLOB-значения

Синтаксис

```
read_blob_real([<курсор>]);
```

<курсор> – курсорная переменная.

Описание

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

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

Возвращаемое значение

1) Считанные данные типа REAL.

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

2) Код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Чтение данных типа numeric из BLOB-значения

Синтаксис

```
read_blob_numeric([<курсор>]);
```

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из BLOB-значения заданного BLOB-столбца текущей записи данных типа NUMERIC.

Предполагается, что в BLOB-значении в текущей позиции для чтения находится двоичное представление данных типа NUMERIC, как описано в функции add_blob.

Возвращаемое значение

1) Считанные данные типа NUMERIC.
2) Код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Чтение данных типа DOUBLE из BLOB-значения

Синтаксис

```
read_blob_double([<курсор>]);
```

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из BLOB-значения заданного BLOB-столбца текущей записи данных типа DOUBLE.

Предполагается, что в BLOB-значении в текущей позиции для чтения находится двоичное представление данных типа DOUBLE, как описано в функции add_blob.

Возвращаемое значение

1) Считанные данные типа DOUBLE.
2) Код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Чтение данных типа CHAR из BLOB-значения

Синтаксис

```
read_blob_char([<курсор>]);
```

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из BLOB-значения заданного BLOB-столбца текущей записи данных типа CHAR.

Предполагается, что в BLOB-значении в текущей позиции для чтения находится двоичное представление данных типа CHAR, как описано в функции add_blob.

Возвращаемое значение

1) Считанные данные типа CHAR.
2) NULL-значение, если BLOB-поле содержит не символьные данные.
3) Код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Чтение данных типа NCHAR из BLOB-значения

Синтаксис

```
read_blob_nchar([<курсор>]);
```

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из BLOB-значения заданного BLOB-столбца текущей записи данных типа NCHAR.

Предполагается, что в BLOB-значении в текущей позиции для чтения находится двоичное представление данных типа NCHAR, как описано в функции add_blob.

Возвращаемое значение

1) Считанные данные типа NCHAR.
2) Код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Чтение данных типа DATE из BLOB-значения

Синтаксис

```
read_blob_date([<курсор>]);
```

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из BLOB-значения заданного BLOB-столбца текущей записи данных типа DATE.

Предполагается, что в BLOB-значении в текущей позиции для чтения находится двоичное представление данных типа DATE, как описано в функции add_blob.

Возвращаемое значение

1) Считанные данные типа DATE.

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

2) Код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Чтение данных типа bool из BLOB-значения

Синтаксис

```
read_blob_bool([<курсор>]);
```

<курсор> – курсорная переменная.

Описание

Функция предназначена для чтения из BLOB-значения заданного BLOB-столбца текущей записи данных типа BOOL.

Предполагается, что в BLOB-значении в текущей позиции для чтения находится двоичное представление данных типа BOOL, как описано в функции add_blob.

Возвращаемое значение

1) Считанные данные типа BOOL.
2) Код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Установка текущей позиции для чтения BLOB-данных

Синтаксис

```
seek_blob([<курсор>,] <позиция>);
```

<курсор> – курсорная переменная;

<позиция> – целочисленное положительное выражение.

Описание

Функция задает позицию для чтения данных из установленного BLOB-столбца текущей записи, то есть 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

1) Значение параметра <позиция>.

Получение размера BLOB-значения

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 записи, содержащей BLOB-значение.

Возвращаемое значение

1) Размер BLOB-значения (тип `INT`).
2) Код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Удаление BLOB-значения

Синтаксис

```
clear_blob([<объект>]);
```

Описание

Функция удаляет BLOB-значение указанного <объекта>, устанавливая его длину в 0. В качестве <объекта> может быть задано BLOB-значение курсора или BLOB-переменная. Удаляемое BLOB-значение курсора должно быть предварительно установлено с помощью функции set_cur_blob, в противном случае исполнение clear_blob() будет иметь неопределенное поведение.

Возвращаемое значение

1) 0 при успешном завершении (значение типа `INT`).
2) NULL при ошибке:
   - `BADPARAM`: неверный параметр функции;
   - `NOMEM`: недостаточно оперативной памяти.
3) Код завершения СУБД ЛИНТЕР, который может быть получен с помощью функции errcode().

Примеры

```
create or replace table D1 (id int, D blob, F blob);
insert into D1 values (1, 'pipe', 'Ivanova');
insert into D1 values (2, 'corner', 'Petrov');
insert into D1 values (3, 'channel', 'Sidorov');

1)
create or replace procedure PRC_CLEAR(in i int) result integer for
 debug
declare
  var b typeof(D1);//
code
  open b for "select * from D1 where id = ?;", i;//
  set_cur_blob(b,2);//
  CLEAR_BLOB(b);//
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
  return errcode();//
end;
select lenblob(D) from D1 where id = 1;
|4|
call PRC_CLEAR(1);
select lenblob(D) from D1 where id = 1;
|0|

2)
create or replace procedure PRC_CLEAR2(in i int; in j int) result
 integer for debug
code
  execute "select * from D1 where id = ?;", i;//
  set_cur_blob(3);//
  clear_blob();//
  return errcode();//
end;
select lenblob(F) from D1 where id = 1;
|7|
call PRC_CLEAR2(1);
select lenblob(F) from D1 where id = 1;
|0|
```

Типизация BLOB-значения

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 INTEGER, 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 "create or replace table tblob(i int, b1 blob, b2
 blob);"; //
  while i <= n loop
    execute direct "insert into tblob(i,b1,b2) values(?, NULL,
 NULL);" using i; //
    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;
```

Длина BLOB-значения

Синтаксис

`length(<BLOB-значение>)`

`<BLOB-значение>` – переменная или поле типа BLOB.

Описание

Вычисляет размер `<BLOB-значения>` в байтах.

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Возвращаемое значение

1) Числовое значение – размер <BLOB-значения> в байтах.
2) Тип возвращаемого значения – INT.

Работа с пакетом данных BSON-формата

BSON (Binary JS Object Notation) – двоичное представление текстового формата документов JSON-формата (*JavaScript Object Notation* – текстовый формат обмена данными, основанный на JavaScript), хранящее данные в виде пар «ключ/значение» как единый объект.

BSON-формат данных применяется для обмена данными с различными приложениями и получения данных из различных источников данных. Данные BSON-пакета обрабатывается как курсор без описателя полей.

Исходный BSON-пакет данных формирует клиентское приложение или получает его готовым из внешнего источника данных (максимальный размер пакета составляет 4000 байт) и передает для дальнейшей обработки СУБД ЛИНТЕР с помощью вызова хранимой процедуры. Хранимая процедура с помощью специальных средств процедурного языка для обработки данных BSON-пакета извлекает из пакета необходимую информацию и выполняет её дальнейшую обработку на стороне ЛИНТЕР-сервера (размещает в БД или в файле, возвращает клиентскому приложению полученный из внешнего источника и преобразованный BSON-пакет и т.п.).

Пример передачи на обработку процедуре pp1 BSON-пакета b1:

```
create or replace procedure pp2( ) result varchar(1024)
declare
  var b1 varbyte(4000);
  var res varchar(1024);
code
  b1 := hextoraw(
"52000000030044000001069002B020000096474 00E0325E7618000000027374720017
000000D092D181D0B5D0BC20D0BFD180D0B8D0B2D0B5D18221000530000204000000050
607080003000500000000" );
  call pp1( b1 ) into res;
  return res;
end;
```

Для извлечения информации из документов BSON-пакета хранимая процедура должна обладать полной информацией об именах и/или типах данных полей документов. Доступ к полям документа возможен как по имени (если поле именованное), так и по их позиционному местоположению (для именованных и не именованных полей).

Схема обработки данных BSON-пакета

Пусть пакет данных BSON имеет следующую структуру:

```
BSON-пакет
'doc1' (int) 'I' (string) 'S'
'doc2' (string) 'S' (doc) 'doc21' (doc) 'doc22'
 .........
'docN' (int) 'I'
```

Для доступа ко всем документам пакета:

1) открыть доступ к пакету BSON:

```
OPEN <cursor_bson> FOR BSON <пакет данных>;
```

2) с помощью оператора FETCH переместиться (позиционироваться) на нужный документ пакета (путем последовательного перемещения по документам или прямым переходом к нужному документу), например, на документ с именем 'doc2':

```
FETCH <cursor_bson> 'doc2';
```

В результате <cursor_bson> будет позиционирован на нужном документе пакета.

3) для перемещения по полям выбранного вложенного документа 'doc2' создать соответствующий курсор для позиционированного на предыдущем шаге документа:

```
OPEN <cursor_doc2> FOR <cursor_bson>;
```

4) с помощью оператора FETCH переместиться (позиционироваться) на нужное поле документа (путем последовательного перемещения по полям документ или прямым переходом к нужному полю), например:

- на вложенный документ с именем 'doc21':

```
FETCH <cursor_doc2> 'doc21';
```

В результате <cursor_doc2> будет позиционирован на нужном поле 'doc21';

- или с помощью функции extract<тип> извлечь значение поля известного типа данных (с помощью функции extractvalue – текстовое значение поля с любым типом данных), например, значение поля 'S':

```
extractstring(<cursor_doc2>, 'S');
```

5) так как поле 'doc21' является вложенным документом, то для доступа к его полям создать соответствующий курсор для позиционированного на предыдущем шаге поля:

```
OPEN <cursor_doc21> FOR <cursor_doc2>;
```

6) с помощью оператора FETCH перемещаться по полям документа с именем 'doc21':

```
FETCH <cursor_doc2> 'doc21';
```

и с помощью функции extract<тип> или extractvalue извлекать значения нужных полей документа.

Доступ к документам пакета

Назначение

Связывание курсорной переменной с пакетом данных формата BSON.

Синтаксис

```
OPEN <курсор> FOR BSON (<пакет данных>);
```

<курсор>::= переменная типа CURSOR;

<пакет данных>::= переменная типа BYTE или VARBYTE.

Описание

<Пакет данных> должен содержать шестнадцатеричные данные в BSON-формате. Максимальная длина пакета 4000 байт.

Оператор выполняет следующие действия:

1) принимает переданный пакет данных и сохраняет его во внутренней рабочей области исполнительной системы процедурного языка СУБД ЛИНТЕР;
2) считывает первую запись (документ) полученного пакета данных и делает её текущей записью курсора;
3) позволяет с помощью оператора FETCH перемещаться по документам пакета;
4) обеспечивает извлечение значений полей документов с помощью специальных функций.

Возвращаемое значение

1) Отсутствует.

Исключения

BADPARAM Нарушение структуры BSON-пакета.

Пример

```
…
declare
  var cur cursor;
  var pack_bson varbyte(4000);
…
! заполнение буфера pack_bson данными BSON-формата
  open cur for bson(pack_bson);
…
```

Доступ к полям документа

Назначение

Связывание курсорной переменной с документом пакета данных формата BSON.

Синтаксис

```
OPEN <курсор> FOR <документ>;
```

<курсор>::= переменная типа CURSOR;

<документ>::= переменная типа CURSOR.

Описание

Аргумент <курсор> задает курсорную переменную, используемую для работы с выбранным документом.

Если предполагается одновременная работа с несколькими документами, должно использоваться соответствующее количество разных курсорных переменных (своя курсорная переменная для каждого документа). Повторное использование одной и той же курсорной переменной возможно только после закрытия связанного с ней курсора (т.е. после освобождения курсорной переменной).

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Аргумент <документ> должен ссылаться на курсор, позиционированный на обрабатываемый документ. Позиционирование (перемещение к нужному документу BSON-пакета) выполняется с помощью оператора FETCH.

Возвращаемое значение

1) Отсутствует.

Исключения

CURNOTOPEN Обращение к неоткрытому курсору.

BADPARAM Нарушение структуры BSON-пакета.

Перемещение по документам пакета

Назначение

Позиционирование курсора в заданное местоположение.

Синтаксис

```
FETCH <курсор> [<ориентация>] [<ключ>];
```

<курсор>::= имя курсорной переменной;

<ориентация>::= FIRST | NEXT;

<ключ>::=символьный англоязычный литерал в кодировке ANSI.

Описание

Имя курсорной переменной в аргументе <курсор> должно ссылаться на ранее открытый курсор BSON-пакета или курсор вложенного документа.

Аргумент <ориентация> задает порядок перемещения по документам пакета или полям документа: FIRST – переход к первому документу пакета (полю документа), NEXT – к следующему (относительно текущего) документу пакета или полю документа. Если аргумент не задан, по умолчанию применяется NEXT.

Аргумент <ключ> задает имя документа пакета или поля документа, к которому должен быть перемещен (позиционирован) курсор. Если пакет или документ содержит несколько документов (полей) с одинаковыми именами, то при наличии аргумента FIRST курсор позиционируется на самом первом документе пакета (поле документа) с указанным именем, если NEXT – то к ближайшему (от текущего) документу пакета (полю документа) с указанным именем.

Если <ключ> не указан, то позиционирование производится на следующий (от текущего) документ пакета (поле документа) независимо от его имени.

Если документ пакета (поле документа) не найден (не найдено), производится позиционирование за пределы пакета документов (после последнего документа) или за пределы полей документа.

Стандартная функция outofcursor() может применяться для работы с курсорами BSON-пакета.

Возвращаемое значение

1) Отсутствует.

Исключения

CURNOTOPEN	Обращение к неоткрытому курсору.
BADPARAM	Нарушение структуры BSON-пакета.

Пример

```
declare
  var c1 cursor;
  var bson varbyte(4000);

  open c1 for bson(b1);
! позиционирование на первый документ с именем «Sale»
  fetch c1 'Sale';
! позиционирование к следующему документу с именем «Sale»
  fetch c1 'Sale';
! позиционирование к следующему документу в пакете
  fetch c1 next;
! возврат к первому документу пакета
  fetch c1 first;
…
  close c1;
```

Извлечение информации из поля документа

Назначение

Предоставление типизированного значения заданного поля текущего документа BSON-пакета.

Синтаксис

extractbool(<курсор>[, <поле>])	Извлекает значение поля типа BOOL (код BSON 0x08)
extractdouble(<курсор>[, <поле>])	Извлекает значение поля типа DOUBLE (код BSON 0x01)
extractint(<курсор>[, <поле>])	Извлекает значение поля типа INTEGER (код BSON 0x10)
extractbigint(<курсор>[, <поле>])	Извлекает значение поля типа BIGINT (код BSON 0x12)
extractstring(<курсор>[, <поле>])	Извлекает значение поля типа STRING (код BSON 0x02)
extractbytes(<курсор>[, <поле>])	Извлекает значение поля типа BYTE (код BSON 0x05)
extractdate(<курсор>[, <поле>])	Извлекает значение поля типа DATE (код BSON 0x09)

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

<курсор>::= имя курсорной переменной;

<поле>::=символьный англоязычный литерал в кодировке ANSI.

Описание

Функции возвращают значение типа данных, извлекаемое из указанного поля в заданном курсоре и соответствующее семантике функции (таблица 6).

Аргумент <поле> задает имя поля, значение которого должно быть получено, в документе, на котором позиционирован курсор. Тип данных поля должен строго соответствовать по семантике используемой для извлечения значения функции (иначе генерируется исключение). Если аргумент <поле> не задан, то при вызове функции значение извлекается из того поля, на котором позиционирован курсор.

Если документ содержит несколько однотипных полей с одинаковыми именами, то значение извлекается из первого поля с указанным именем в заданном курсоре.

Возвращаемое значение

1) Если извлекаемое из поля значение по типу данных совпадает с семантикой функции, то возвращается значение поля, в противном случае генерируется исключение и возвращается NULL-значение.
2) В случае, когда извлекаемое из поля значение является NULL-значением (код BSON 0x0A), все функции возвращают NULL-значение без генерирования исключения.

Таблица 6. Тип данных возвращаемого значения

Функция	Тип данных возвращаемого значения
extractbool	BOOL
extractdouble	DOUBLE
extractint	INTEGER
extractbigint	BIGINT
extractstring	VARCHAR
extractbytes	BYTE
extractdate	DATE

Исключения

CURNOTOPEN	Обращение к неоткрытому курсору.
BADPARAM	Нарушение структуры BSON-пакета.
ERRTYPOPERAND	Несоответствие типа операнда.

Пример

```
{
  "i" : (int)"555",
  "dt" : (date)"01/05/1973 00:45:00",
  "str" : "Всем привет!",
  "0" : { 0x04 05 06 07 08 }
},
```

```
{
}

create or replace procedure pp1( in b1 varbyte( 4000 ) ) result
 varchar(1024)
declare
  var c1 cursor;
  var res varchar(128);
  var resdt date;
code
  res := "";
  open c1 for bson( b1 );
  res := res + tochar( extractbytes( c1, "0" ) ) + " ";
  res := res + tochar( extractint( c1, "i" ) ) + " ";
  res := res + tochar( extractdate( c1, "dt" )  );
  res := res + tochar( extractstring( c1, "str" )  );
  fetch c1 next;
  if outofcursor( c1 ) then
    res := "";
  endif;
  close c1;
  return res;
end;
```

Извлечение информации из вложенного документа

Назначение

Предоставление символьного значения заданного поля вложенного документа.

Синтаксис

```
extractvalue (<курсор>[,<поле>]);
```

<курсор>::= имя курсорной переменной;

<поле>::=символьный англоязычный литерал в кодировке ASCII.

Описание

В случае указания параметра <поле> имя курсорной переменной в аргументе <курсор> должно ссылаться на позиционированный в заданном курсоре вложенный документ.

Аргумент <поле> должен задавать имя поля во вложенном документе (если поле, на котором позиционирован курсор, не является вложенным документом, генерируется исключение).

Если аргумент <поле> задан, осуществляется поиск первого поля с таким именем во вложенном документе.

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Если аргумент <поле> не задан, возвращается значение текущего поля, на котором позиционирован курсор.

Возвращаемое значение

1) Возвращается значение типа VARCHAR, полученное в результате извлечения значения указанного поля из вложенного документа.
2) Если поле с заданным именем не найдено, курсор не открыт или не позиционирован на поле типа «документ», то генерируется исключение и возвращается NULL-значение.

Пример

```
create or replace procedure pp1( in b1 varbyte( 4000 ) ) result
 varchar(1024)
declare
  var c1, c2 cursor;
  var res varchar(128);
  var resdt date;
code
  res := "";
  open c1 for bson( b1 );
  while not outofcursor( c1 ) loop
    open c2 for c1;
    while not outofcursor( c2 ) loop
      res := res + extractvalue( c2 ) + " ";
      fetch c2 next;
    endloop;
    close c2;
    fetch c1 next;
  endloop;
  close c1;
  return res;
end;
```

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 литералы 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, то есть 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 (то есть 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 «?»).

Функции

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).

Если аргумент (аргументы) функции заданы неверно, возвращается NULL-значение.

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

1) <Строка>, дополненная слева указанными последовательностями символов.
2) Тип возвращаемого значения совпадает с типом аргумента.
3) 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

1) <Строка>, дополненная справа указанными последовательностями символов.
2) Тип возвращаемого значения совпадает с типом аргумента.
3) Если <строка> имеет NULL-значение, возвращается NULL-значение.

Примеры

```
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
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Получение 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 <длину>. Если указана слишком большая длина, возвращаются все символы до конца исходной строки.

Возвращаемое значение

1) Тип возвращаемого значения совпадает с типом первого аргумента.
2) Если <строка> имеет NULL-значение или является пустой, возвращается, соответственно, NULL-значение или пустая строка, независимо от остальных параметров.
3) При указании <нач поз> менее 1 или <длины> меньше 0, возвращается NULL-значение и генерируется исключение BADPARAM.

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);         // ""
```

Получение правой части строки

Синтаксис

```
right_substr(<строка>,<количество>)
```

<строка> – выражение типа CHAR, VARCHAR;

<Количество> – целое положительное число.

Описание

Выделяет правую часть <строки> размером <количество>.

Возвращаемое значение

1) Тип возвращаемого значения совпадает с типом аргумента.
2) Если <строка> имеет NULL-значение, возвращается NULL-значение.

Примеры

```
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 aa… a.a…(n) в формат a.a…nnnnn
 (например, 234.56(9) в 234.5699999,  .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

1) Строка, являющаяся конкатенацией исходной <строки> заданное <количество> раз.

Примеры

```
1)
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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

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

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

1) 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.
2) Информация о недопустимых значениях входных параметров не возвращается.
3) Тип возвращаемого значения – INT.
4) Если <строка> имеет NULL-значение, результат будет NULL.

Пример

```
create or replace procedure sp_test_instr() result int
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
  return i;
end;
Результат: 24.
```

Длина символьной строки

Синтаксис

```
len|length|char_length(<строка>)
```

<строка> – выражение типа CHAR, VARCHAR.

Описание

Вычисляет длину символьной строки.

Возвращаемое значение

1) Числовое значение – длина <строки> в символах.
2) Если <строка> имеет NULL-значение, возвращается NULL-значение.
3) Тип возвращаемого значения – INT.

Примеры

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

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

3)
line1:="abcd";
line2:="12345";
i:=char_length(line1+line2); // 9
```

Длина байтовой строки

Синтаксис

```
octet_length(<строка>)
```

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

Описание

Определение длины строки в байтах.

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Возвращаемое значение

1) Если <строка> имеет тип данных CHAR, VARCHAR, то возвращается то же значение, что и для функции len (за исключением использования кодировок, в которых символ может быть представлен более чем одним байтом).
2) Если <строка> имеет тип данных NCHAR, NVARCHAR, то возвращаемое значение равно L*2, где L – длина <строки> в символах.
3) Тип возвращаемого значения – INT.
4) Если <строка> имеет NULL-значение, возвращается NULL-значение.

Примеры

```
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.

Возвращаемое значение

1) <Строка> с удаленными слева и справа пробелами.
2) Тип возвращаемого значения совпадает с типом аргумента.
3) Если аргумент имеет NULL-значение, результат будет NULL-значение.

Пример

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

Удаление левосторонних символов

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> становится пустой: то есть длина строки станет равной 0.

Возвращаемое значение

1) <Строка> с удаленными слева указанными символами.
2) Тип возвращаемого значения совпадает с типом аргумента.
3) Если один из аргументов имеет NULL-значение, результат будет NULL-значение.

Примеры

```
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); // 0
```

Удаление правосторонних символов

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> становится пустой: то есть длина строки станет равной 0.

Возвращаемое значение

1) <Строка> с удаленными справа указанными символами.

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

2) Тип возвращаемого значения совпадает с типом аргумента.

3) Если один из аргументов имеет NULL-значение, результатом будет NULL-значение.

Примеры

```
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); // 0
```

Поиск 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.

Примеры

```
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>)
overlay(<строка>, <подстрока>, <позиция>[, <длина>])
```

<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>.

Если <длина> не задана (в функции overlay), используется длина <подстроки>.

<Подстрока> задает значение, вставляемое вместо удаленной подстроки.

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

Количество заменяемых символов может превышать количество удаляемых.

Возвращаемое значение

1) <Строка> с замененной <подстрокой>.
2) Код завершения при неправильных значениях аргументов функции.
3) Если <строка> имеет значение NULL, результат будет NULL.

Примеры

```
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";
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
5)
create or replace procedure tst_overlay() result int for debug
declare
  var line char(50);
code
  line:=overlay("1234567890","abc", 4, 3); -- 123abc7890
  line:=overlay("1234567890","abcde", 6, 3); --12345abcde90
  line:=overlay("1234567890","abcde", 6, 5); -- 12345abcde
  line:=overlay("1234567890","abc", 10); -- 123456789abc
  line:=overlay("1234567890","abc",2,6); -- 1abc890
  return 0;
end;
```

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

1) Исходная <строка>, в которой все вхождения <подстроки 1> заменены на <подстроку 2>.
2) Если значение <подстроки 1> в <строке> не найдено, <строка> возвращается без изменений.
3) Если <строка> имеет NULL-значение, результат будет NULL-значение.

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.



Примечание
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

1) 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'.
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>.
3) Если один из аргументов имеет NULL-значение, результат будет NULL-значение.

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>[, ... ])
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

<строка> – выражение типа CHAR, VARCHAR.

Описание

<Строка> может содержать знаки вопроса «?».

Возвращаемое значение

1) Строка, в которой вместо знаков вопроса подставлены значения параметров, преобразованные к строке. Соответствие знаков вопроса и параметров устанавливается по порядку.
2) Если необходимо включить в строку сам символ вопроса, он экранируется при помощи двойного или тройного обратного слеша (\\? или \\\?). Экранирование выполняется в 2 этапа:

   а) удаление экранирующего символа:

   - любая строка в процедуре проверяется на наличие обратных слешей: в результате этой проверки производится преобразование;
   - строки вида "\?" в строку вида "?";
   - строки вида "\\?" в строку вида "\?";
   - строки вида "\\\?" в строку вида "\?";
   - строки вида "\\\\?" в строку вида "\\?" и т.д.

   б) получившаяся строка проверяется на наличие "\?".
3) Если количество вопросов и актуальных выражений-параметров не совпадает, возвращается NULL.

Примеры

```
1) 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».
2) Включение в таблицу tab_mkstr(i int, quest char(20)) строки с
 символом вопроса:
execute direct makestr("insert into ? values(?, '\\?');",
 "tab_mkstr", 2); //
execute direct makestr("insert into ? values(?, '\\\?');",
 "tab_mkstr", 3); //
Результат: выполнения примера:
   I           QUEST
```

```
  -            -----
 |           2|?                    |
 |           3|?                    |
```

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

<Строка>, в которой каждое вхождение <символа> удвоено.

Преобразование байтовой строки в символьную

Синтаксис

chr(<строка>)

<строка> – значение типа BYTE.

Возвращаемое значение

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

Преобразование символьной строки в байтовую

Синтаксис

asc(<строка1>,<строка2>)

<строка1> – значение типа CHAR или типа NCHAR;

<строка2> – значение типа BYTE.

Возвращаемое значение

1) 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.
2) 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:

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
asc(unic_var, out);
execute "insert into t(uc) values (?);" using hex(btoa(out));
```

Числовое представление символа

Синтаксис

```
ascii(<строка>)
```

<строка> – значение типа CHAR, VARCHAR.

Возвращаемое значение

1) Функция возвращает значение типа INTEGER ASCII-кода первого символа <строки>.
2) Если аргумент NULL, результат NULL.

Пример

```
create procedure ascii_test(in arg char(20)) result int
code
  return ascii(arg);
end;

execute ascii_test("abc");
return value = 97

execute ascii_test('1');
return value = 49

execute ascii_test(null);
return value = NULL
```

Преобразование строки к верхнему регистру

Синтаксис

```
toupper|upper(<строка>)
```

<строка> – значение типа CHAR, VARCHAR.

Возвращаемое значение

1) <Строка>, в которой все символы имеют заглавное (прописное) представление, то есть буквы алфавита a-z, а-я преобразованы в A-Z, А-Я.
2) Тип возвращаемого значения совпадает с типом аргумента.
3) Если аргумент NULL-значение, результат NULL-значение.

Преобразование строки к нижнему регистру

Синтаксис

```
tolower|lower(<строка>)
```

<строка> – значение типа CHAR, VARCHAR.

Возвращаемое значение

1) <Строка>, в которой все символы имеют строчное представление, то есть буквы алфавита A-Z, А-Я преобразованы в a-z, а-я.
2) Тип возвращаемого значения совпадает с типом аргумента.
3) Если аргумент NULL-значение, результат NULL-значение.

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

1) Строка того же типа и длины.
2) Если аргумент является NULL-значением, результат NULL-значение.

Примеры

```
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
```

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>][<$>][{<символ>[.<символ>]} | {<цифра>[.<цифра>]EEEE}]

<символ>::= <цифра> | , | X

<цифра>::= '0'|'9'
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

'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 '$'

. – десятичная 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 "insert into test_to_char_res values(?, ?);" using 7B,
c;
 c:=to_char(2063597568I, "XXXXXXXXX");
 execute "insert into test_to_char_res values(?, ?);" using
7B000000, c;
 c:=to_char(123B, "XX");
 execute "insert into test_to_char_res values(?, ?);" using 7B,
c;
 c:=to_char(2063597568B, "XXXXXXXXX");
 execute "insert into test_to_char_res values(?, ?);" using
7B000000, c;
 c:=to_char(34621422143503227, "XXXXXXXXXXXXXXX");
 execute "insert into test_to_char_res values(?, ?);" using
7B0000007B7B7B, c;
 c:=to_char(+123.45N, "XX");
 execute "insert into test_to_char_res values(?, ?);" using 7B,
c;
 c:=to_char(+2063597568.45N, "XXXXXXXXX");
 execute "insert into test_to_char_res values(?, ?);" using
7B000000, c;
 c:=to_char(-123.45N, "XXX");
 execute "insert into test_to_char_res values(?, ?);" using ####,
c;
```

```
//  /* EEEE */
  c:=to_char(+123.456N, "9.9EEEE");
  execute "insert into test_to_char_res values(?, ?);" using 1.2E
+02, c;
  c:=to_char(+123.456N, "FM9.9EEEE");
  execute "insert into test_to_char_res values(?, ?);" using 1.2E
+02, c;
  c:=to_char(1000.0D, "FM9.9EEEE"); //+1E+03D
  execute "insert into test_to_char_res values(?, ?);" using 1.E
+03, c;
  c:=to_char(-0.001D, "FM9.9EEEE"); //-1E-03D
  execute "insert into test_to_char_res values(?, ?);" using
 -1.E-03, c;
  c:=to_char(1.1111D, "S$9EEEE");
  execute "insert into test_to_char_res values(?, ?);" using +$1E
+00, c;
  c:=to_char(-1.1111D, "S$9EEEE");
  execute "insert into test_to_char_res values(?, ?);" using -$1E
+00, c;
  c:=to_char(-1.1111D, "FMS$9EEEE");
  execute "insert into test_to_char_res values(?, ?);" using -$1E
+00, c;
  c:=to_char(-1.1121D, "FMS$9.999EEEE");
  execute "insert into test_to_char_res values(?, ?);" using -
$1.112E+00, c;
```

Фонетический код строки

Синтаксис

`soundex(<значение>)`

<значение> – выражение типа CHAR, VARCHAR.

Описание

Получить фонетический код значения.

Возвращаемое значение

1) Символьная строка, представляющая фонетический код заданного <значения>.
2) Тип результата – CHAR(4).
3) Если <значение> имеет NULL-значение, результат будет NULL-значение.

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

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Примеры

```
1) Фонетические коды 0124 указывают на близкое фонетическое
 звучание слов «опечатка» и «опичатка».
new_line:=soundex("опечатка"); // О124
new_line:=soundex("опичатка"); // О124
2) Фонетические коды В700 указывают на близкое фонетическое
 звучание фраз в предложении «Вы ли выли?».
new_line:=soundex("выли"); // В700
new_line:=soundex("вы ли"); // В700
3)
new_line:=soundex("пень"); // П900
new_line:=soundex("лень"); // Л900
```

Сравнение 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

1) Разность фонетического звучания двух аргументов, вычисляемая на основании кодов фонетического значения аргументов (см. функцию soundex).
2) Тип результата – INTEGER.
3) Значение 0 указывает на фонетическое совпадение аргументов, 1 – на существенное различие.
4) Если <значение 1> или <значение 2> имеют NULL-значение, результат будет NULL-значение.

Примеры

```
i:=difference("столб", "столп"); // 0
i:=difference("ошибка", "ашипка"); // 1
```

Символьные UNICODE-функции

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

1) Длина <строки> в UNICODE-символах.
2) 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.

См. описание функции trim.

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.

Левостороннее дополнение UNICODE-строки

Синтаксис

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

<строка> – выражение типа NCHAR, NVARCHAR;

<новая длина> – беззнаковый числовой литерал;

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

<дополняемые символы> – выражение типа NCHAR, NVARCHAR.

Описание

Функция дополняет строку заданными символами с левого края.

См. описание функции lpad.

Правостороннее дополнение UNICODE-строки

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.

Получение правосторонней подстроки 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.

Поиск UNICODE-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 UNICODE-строки к верхнему регистру

Синтаксис

```
ntoupper|nupper(<строка>)
```

<строка> – значение типа NCHAR, NVARCHAR.

Описание

Преобразованное к верхнему регистру значение <строки>.

См. описание функции toupper, upper.

Преобразование UNICODE-строки к нижнему регистру

Синтаксис

```
ntolower|nlower(<строка>)
```

<строка> – значение типа NCHAR, NVARCHAR.

Описание

Преобразованное к нижнему регистру значение <строки>.

См. описание функции tolower, lower.

Преобразование значения в UNICODE-строку

Синтаксис

```
tonchar(<значение> [, <параметры>])
```

<значение> – выражение любого допустимого типа.

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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 в разделе Поддержка кодовых страниц).

Удаление левосторонних символов из UNICODE-строки

Синтаксис

```
nltrim(<строка> [,<подстрока>])
```

<строка> – выражение типа NCHAR, NVARCHAR;

<подстрока> – выражение типа NCHAR, NVARCHAR.

Описание

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

См. описание функции ltrim.

Удаление правосторонних символов из UNICODE-строки

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>)
noverlay(<строка>, <подстрока>, <позиция>[, <длина>])
```

<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 её замена).

См. описание функций insert, overlay.

Дополнительное условие: количество удаляемых символов не должно выходить за пределы строки, т.е. сумма значений <позиция> + <длина> должна быть не больше, чем значение «длина <строки> + 1» (данное условие не действует для символьных типов в версиях 6.X, но действует для функции ninsert).

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.

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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

1) Тип возвращаемого значения совпадает с типом аргумента.
2) 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

1) Результат округления (до целого) с избытком <значимого числового выражения>.
2) Тип возвращаемого значения – DOUBLE.
3) Функция возвращает наименьшее целое значение, большее или равное аргументу.
4) 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

1) Результат округления (до целого) с недостатком <значимого числового выражения>.

2) Тип возвращаемого значения – DOUBLE.

3) 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' | 'DY' | 'DAY' | '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

1) <Выражение>, округленное до заданной точности.
2) Тип возвращаемого результата – DATE.
3) Результат при указании <точность> 'D' зависит от наличия ключа /COMPATIBILITY=ORACLE в команде запуска ядра СУБД:
   - значение DATE, округленное до текущего дня, если ключ не задан;
   - значение DATE, округленное до ближайшего дня начала недели, если ключ задан.
4) Результат при указании <точность> 'DY' или 'DAY' будет округлен до ближайшего дня начала недели.
5) Если аргумент NULL, результат NULL.

Пример

```
create or replace procedure date_round_test(in arg char(5)) result
 date for debug
code
  return date_round(atod("22.08.2015:18:32:55.87"),arg);//
end;
--
call date_round_test('D');
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
call date_round_test('Y');
```

Результат:

```
Return value = 23.08.2015:00:00:00.00
Return value = 01.01.2016:00:00:00.00
```

Усечение представления значения типа «дата-время»

Синтаксис

```
date_trunc(<выражение>,<точность>)
```

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

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

```
{'D' | 'DY' | 'DAY' | '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

1) <Выражение>, усеченное до заданной точности.
2) Тип возвращаемого результата – DATE.
3) Результат при указании <точность> 'D' зависит от наличия ключа /COMPATIBILITY=ORACLE в команде запуска ядра СУБД:
   - значение DATE, усеченное до текущего дня, если ключ не задан;
   - значение DATE, усеченное до ближайшего дня начала недели, если ключ задан.
4) Результат при указании <точность> 'DY' или 'DAY' будет усечен до ближайшего дня начала недели.
5) Если аргумент <выражение> NULL, результат NULL.

Пример

```
create or replace procedure date_trunc_test(in arg char(5)) result
 date for debug
code
  return date_trunc(atod("22.08.2015:18:32:55.87"),arg);//
end;
--
call date_trunc_test('D');
call date_trunc_test('Y');
```

Результат:

```
Return value = 22.08.2015:00:00:00.00
```

```
Return value = 01.01.2015:00:00:00.00
```

Тригонометрические функции

Синтаксис

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

<значимое числовое выражение> – угол, выраженный в радианах в пределах от -2^{63} до $+2^{63}$.

Описание

Вычисление тригонометрических функций.

Возвращаемое значение

1) Значение косинуса (COS), синуса (SIN) и тангенса (TAN) <значимого числового выражения> в радианах.
2) Тип возвращаемого результата – DOUBLE.
3) 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

1) Значение обратного косинуса (ACOS), обратного синуса (ASIN), обратного тангенса (ATAN) от <значимого числового выражения> в радианах и значение обратного тангенса (ATAN2) от значения <значимое числовое выражение 1>/<значимое числовое выражение 2>.
2) Тип возвращаемого результата – DOUBLE.
3) Если аргумент NULL, результат NULL.

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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

1) Значение гиперболического косинуса (COSH), синуса (SINH) и тангенса (TANH) <значимого числового выражения> в радианах.
2) Тип возвращаемого результата – DOUBLE.
3) Если аргумент NULL, результат NULL.

Экспоненциальная функция

Синтаксис

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

Описание

Вычисление экспоненциального значения.

Возвращаемое значение

1) Экспоненциальное значение <значимого числового выражения>.
2) Тип возвращаемого результата – DOUBLE.
3) Если аргумент NULL, результат NULL.

Логарифмические функции

Синтаксис

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

<основание> ::= вещественный литерал.

Описание

Вычисление логарифмических функций.

Возвращаемое значение

1) Функция ln возвращает натуральный логарифм <значимого числового выражения>.

2) Функция log возвращает логарифм <значимого числового выражения> по заданному <основанию>.

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

4) Тип возвращаемого результата в обоих случаях – DOUBLE.

5) 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

1) <Значимое числовое выражение>, округленное до заданной точности;

2) 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)).

3) Тип возвращаемого результата – DOUBLE.

4) 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).

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Отрицательное значение – точность округления перед десятичной точкой (то есть при -1 до ближайшего целого десятка, -2 – до сотни, -3 до тысячи и так далее).

Возвращаемое значение

1) <Значимое числовое выражение>, округленное до заданной точности.
2) Тип возвращаемого результата – DOUBLE.
3) Если аргумент NULL, результат NULL.

Определение знака числа

Синтаксис

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

Описание

Определение знака числа.

Возвращаемое значение

1) -1 – <значимое числовое выражение> отрицательное.
2) 1 – <значимое числовое выражение> положительное.
3) 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

1) Корень квадратный из <значимого числового выражения>.
2) Тип возвращаемого результата – DOUBLE.
3) Если аргумент отрицательный, вызывается исключение BADPARAM.
4) Если аргумент NULL, результат NULL.

Календарные функции

Выделение дня из даты

Синтаксис

```
day(<дата>)
```

<дата> – выражение типа DATE.

Описание

Возвращает номер дня из <даты>.

Возвращаемое значение

Тип возвращаемого значения – INT.

Пример

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

Выделение месяца из даты

Синтаксис

```
month(<дата>)
```

<дата> – выражение типа DATE.

Описание

Возвращает номер месяца из <даты>.

Возвращаемое значение

Тип возвращаемого значения – INT.

Пример

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

Выделение года из даты

Синтаксис

```
year(<дата>)
```

<дата> – выражение типа DATE.

Описание

Возвращает номер года из <даты>.

Возвращаемое значение

Тип возвращаемого значения – INT.

Пример

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

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Выделение часа из даты

Синтаксис

```
hour(<дата>)
```

<дата> – выражение типа DATE.

Описание

Возвращает номер часа из <даты>.

Возвращаемое значение

Тип возвращаемого значения – INT.

Пример

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

Выделение минут из даты

Синтаксис

```
minute(<дата>)
```

<дата> – выражение типа DATE.

Описание

Возвращает номер минуты из <даты>.

Возвращаемое значение

Тип возвращаемого значения – INT.

Пример

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

Выделение секунд из даты

Синтаксис

```
second(<дата>)
```

<дата> – выражение типа DATE.

Описание

Возвращает номер секунды из <даты>.

Возвращаемое значение

Тип возвращаемого значения – INT.

Пример

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

Выделение тиков из даты

Синтаксис

```
ticks(<дата>)
```

<дата> – выражение типа DATE.

Описание

Возвращает номер тика из <даты>.

Возвращаемое значение

Тип возвращаемого значения – INT.

Пример

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

Формирование даты

Синтаксис

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

<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 по Гринвичу

Синтаксис

```
sysdate()
```

Описание

Получение текущих даты и времени по Гринвичу.

Возвращаемое значение

Текущие дата и время по Гринвичу.

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Получение текущей локальной даты

Синтаксис

```
localdate()
```

Описание

Получение текущих локальных даты и времени.

Возвращаемое значение

Текущие локальные дата и время 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

1) Значение типа DATE в полном формате по умолчанию, представляющее дату последнего дня того месяца, который выбран из аргумента функции.
2) Если <значимое выражение> содержит дату в неполном формате, на места недостающих значений подставляются нули.

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 (таблица 7).

Таблица 7. Значение <дня недели>

Значение <дня недели>		Соответствующий день недели
полное	**сокращенное**	
Monday	Mon	Понедельник
Tuesday	Tue	Вторник

Значение <дня недели>		Соответствующий день недели
полное	сокращенное	
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 (то есть через 7 дней) дня недели.

Возвращаемое значение

1) Значение типа DATE в полном формате по умолчанию, соответствующее указанному <дню недели> после заданной даты.
2) Если <значимое выражение> содержит дату в неполном формате, на места недостающих значений подставляются нули.

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.

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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

Возвращаемое значение

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> (регистронезависимый).

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.

Допустимые значения <параметра> приведены в таблице 8.

Таблица 8. Соответствие <параметра> и возвращаемого функцией datesplit значения

Значение <параметра>	Возвращаемое значение
"D"	День месяца
"M"	Номер месяца
"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

Значение <параметра>	Возвращаемое значение
"SS"	Количество секунд
"FF"	Количество тиков

Возвращаемое значение

1) Указанный элемент <значимого выражения>.
2) Тип возвращаемого значения – INT.

Примеры

```
// 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 <типа интервала> приведены в таблице 9.

Таблица 9. Допустимые значения <типа интервала> функции multime

Значение <типа интервала>	Интервал времени
1	Тики
2	Секунды
4	Минуты
8	Часы
16	Дни
32	Недели
64	Месяцы
128	Кварталы
256	Годы

Примечание
Значение <интервала> не должно задавать дату более 9999 года.

Возвращаемое значение

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

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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

Примеры

```
// 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 multime.

Возвращаемое значение

1) Значение типа INT, представляющее разницу между конечной и начальной датами в единицах измерения, заданных параметром <тип интервала>.
2) 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 (минус 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 оно приводится к целому числу (к типу INTEGER).

Для NULL-значения возвращается строка «NULL».

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'
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
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, day, Day, DAY – день (day – название дня недели);
- MM, mm, mon, Mon, MON – месяц (mon – название месяца);
- YY, YYYY – год;
- HH, HH12, 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"
"Day-Mon-YY"
"Day-Mon-YYYY"
"mm/day/yy",
"day.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> равен FALSE;
- в формате DD.MM.YYYY:HH:MI:SS.FF, если параметр <формат представления> равен TRUE или не задан;
- в заданном формате, если параметр <формат представления> является строковым литералом.

Для NULL-значения возвращается строка «NULL».

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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 с SQL-функцией `hex`), иначе значения разделяются пробелами.

Для NULL-значения возвращается строка «NULL».

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.[YY]YY[:HH[:MI[:SS[.FF]]]].

Допустимые <форматы представления> см. в описании функции dtoa.

Если параметр содержит неверное представление даты, возвращается NULL-значение.

Возвращаемое значение

1) Тип возвращаемого значения – DATE.
2) При ошибке преобразования возвращается начальная дата.
3) Для NULL-значения возвращается строка «NULL».

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
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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 функции выходит за верхний предел допустимого диапазона (+32 767), то возвращается значение верхнего предела, если же значение параметра выходит за нижний предел диапазона (-32 768), то возвращается значение нижнего предела.

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.

Преобразование 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 функции выходит за верхний предел допустимого диапазона (+2 147 483 647), то возвращается значение верхнего предела, если же значение параметра выходит за нижний предел диапазона (-2 147 483 648), то возвращается значение нижнего предела.

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 функции 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.

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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 numeric, 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 CHAR, VARCHAR, содержащая шестнадцатеричные цифры (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 запись за пределами выборки, иначе – FALSE. Выбор за пределами происходит, если выполняется FETCH NEXT на последней записи, FETCH PREVIOUS на первой или в результате FETCH ABSOLUTE/FETCH RELATIVE, если осуществляется запрос на запись с несуществующим номером.

Пример

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

Количество записей в курсорной выборке, сделанной по курсору

Синтаксис

```
rowcount(<курсор>)
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

<курсор> – курсорная переменная.

Описание

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

Определение кода завершения SQL-команды

Синтаксис

```
errcode([<курсор>])
```

<курсор> – имя курсорной переменной.

Описание

Возвращает значение типа INTEGER – код завершения, возникший при выполнении SQL-команды по указанному курсору. Если курсорная переменная не задана, возвращается код завершения для последнего выполненного оператора EXECUTE.

Пример

```
create or replace procedure sp_errcode() result int for debug
declare
  var i int;
code
  execute "select * from auto where personid = -1";
exceptions
  when all then return errcode();
end;
call sp_errcode();
Результат 2
```

Определение сообщения завершения SQL-команды

Синтаксис

```
sqlerrm([<код завершения>])
```

<код завершения> – код завершения ядра СУБД.

Описание

Возвращает значение типа CHAR(4000) – сообщение завершения для указанного кода завершения ядра СУБД. Если код завершения не указан, то будет возвращено сообщение для последнего выполненного оператора EXECUTE по курсору по умолчанию.

Пример

```
create or replace procedure sp_sqlerrm() result char(100) for
 debug
code
```

```
  execute direct "select * from xxxxx";
exceptions
  when all then return sqlerrm();
end;
call sp_sqlerrm();
Результат несуществующая таблица
```

Определение кода исключения

Синтаксис

```
exccode()
```

Описание

Возвращает значение типа `INTEGER` – код исключения процедурного языка.

Значение кода исключения:

1) если исключение возникло при выполнении SQL-команды, то будет возвращен код завершения для последнего выполненного оператора `EXECUTE` по курсору;
2) если исключение возникло при выполнении оператора процедурного языка, то будет возвращен код исключения процедурного языка (см. Виды критичных исключений).

Пример

```
1) получение кода исключения процедурного языка при выполнении
 операторов процедурного языка:
create or replace procedure sp_exccode() result int for debug
declare
  var i int;
code
  i := 1/0;
exceptions
  when all then return exccode();
end;
call sp_exccode();
Результат -2
2) получение кода исключения при выполнении SQL-команды по курсору
 по умолчанию:
create or replace procedure sp_exccode2() result int for debug
declare
  var i int;
code
  execute direct "selectt * from xxxxx";
exceptions
  when all then return exccode();
end;
call sp_exccode2();
Результат 2051
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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.

После удаления текущей строки курсор автоматически встает на следующую строку, если ее нет – на предыдущую. Соответственно, оператор FETCH сдвигает курсор еще на одну строку. Не произойти изменения текущей строки после операции delete current of cursor никак не может, так как этой текущей строки уже нет. Номер текущей строки по возможности сохраняется.

Пример

В приведенном примере удаление строк выполняется не подряд, через одну строку:

```
while not outofcursor(a) loop
  execute "delete from tab1 where current of \"cursor_a\";";//
  print ("Текущая строка курсора:"+ to_char(currow(a)));//
  fetch a;//
endloop
```

Прочие функции

Вычисление максимального значения из пары значений

Синтаксис

```
max(<значимое числовое выражение 1>,<значимое числовое выражение
 2>)
```

Описание

Вычисление максимального значения из пары значений.

Возвращаемое значение

1) Максимальное значение из пары <значимое числовое выражение 1>, <значимое числовое выражение 2>.
2) Тип возвращаемого значения устанавливается по типу данных первого аргумента.

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 максимального значения из набора значений

Синтаксис

```
greatest(<значимое выражение 1>,… <значимое выражение n>)
```

```
<значимое выражение>::=<значимое числовое выражение>|<значимое выражение типа «дата-время»>
```

Описание

Вычисление максимального значения из набора значений.

<Значимое выражение> может иметь числовой или «дата-время» тип данных.

Типы данных всех элементов списка должны быть совместимы.

<Значимые выражения> не могут быть NULL-значениями.

Возвращаемое значение

1) Максимальное значение из набора значений <значимое выражение 1>,… <значимое выражение n>.
2) Тип возвращаемого значения устанавливается по типу данных первого аргумента.
3) Если тип данных первого аргумента INT, то для результирующих NUMERIC-значений происходит отбрасывание значений после запятой.

Примеры

```
1)
create or replace procedure tst_greatest_int(in p_1 int; in p_2
 int;) result int
code
  return greatest (p_1, p_2); //
end;
execute tst_greatest_int(1, 2);
Результат 2
2)
create or replace procedure tst_greatest_date() result date
declare
  var d1 date;
  var d2 date;
code
  d1:=atod("28.04.2015","dd.mm.yyyy");
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
  d2:=atod("28.05.2016","dd.mm.yyyy");
  return greatest (D1, D2);
end;

execute tst_greatest_date();
Результат 05/28/2016:00:00:00.00
3)
create or replace procedure tst_greatest_date2(in dt1 date; in dt2
 date) result date
code
  return greatest(dt1, dt2); //
end;
call tst_greatest_date2('20.07.2015','21.07.2015');
Результат 07/21/2015:00:00:00.00
```

Вычисление минимального значения из пары значений

Синтаксис

```
min(<значимое числовое выражение 1>,<значимое числовое выражение
 2>)
```

Описание

Вычисление минимального значения из пары значений.

Возвращаемое значение

1) Минимальное значение из пары <значимое числовое выражение 1>, <значимое числовое выражение 2>.
2) Тип возвращаемого значения устанавливается по типу данных первого аргумента.
3) Если один из аргументов равен NULL-значению, генерируется исключение BADPARAM.

Примеры

```
1) Вычисление минимального значения:
create or replace procedure tst_min(in arg1 int; in arg2 int)
 result int
code
  return min(arg1, arg2); //
end;
2) Вычисление минимального значения из набора значений:
create or replace procedure tst_min(in arg1 int; in arg2 int; in
 arg3 int; in arg4 int) result int
code
  return min(min(min(arg1, arg2), arg3), arg4); //
end;
3) Вычисление максимального значения:
```

```
create or replace procedure tst_min(in arg1 int; in arg2 int)
 result int
code
  if min(arg1, arg2) = arg1 then
    return arg2;//
  else
    return arg1; //
  endif
end;
```

Вычисление минимального значения из набора значений

Синтаксис

```
least(<значимое выражение 1>,… <значимое выражение n>)
```

```
<значимое выражение>::=<значимое числовое выражение>|<значимое выражение типа «дата-время»>
```

Описание

Вычисление минимального значения из набора значений.

<Значимое выражение> может иметь числовой или «дата-время» тип данных.

<Значимые числовые выражения> не могут быть NULL-значениями.

Возвращаемое значение

1) Минимальное значение из набора значений <значимое выражение 1>,… <значимое выражение n>.
2) Тип возвращаемого значения устанавливается по типу данных первого аргумента.
3) Если тип данных первого аргумента INT, то для результирующих NUMERIC-значений происходит отбрасывание значений после запятой.

Примеры

```
1)
create or replace procedure tst_least_int(in arg1 int; in arg2
 int; in arg3 double) result numeric
code
  return least(arg1, arg2, arg3); //
end;
execute tst_least_int(5,2,1.675);
Результат 1.0
2)
create or replace procedure tst_least_date(in dt1 date; in dt2
 date) result date
code
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
  return least(dt1, dt2); //
end;
call tst_least_date('20.07.2015','21.07.2015');
Результат 07/20/2015:00:00:00.00
```

Вычисление остатка от деления

Синтаксис

```
mod(<делимое>, <делитель>)
```

Описание

Вычисление остатка от деления.

Возвращаемое значение

1) Остаток от деления <делимого> на <делитель>, которые могут быть выражениями любого числового типа.
2) Если значение <делителя> равно 0, возникает исключение BADPARAM.
3) Тип возвращаемого результата – DOUBLE.
4) Если какой-нибудь из аргументов NULL, результат NULL.

Генерация псевдослучайного числа

Синтаксис

```
rand()
```

Описание

Получение псевдослучайного числа.

Возвращаемое значение

1) Положительное псевдослучайное целое число в диапазоне от 0 до 2147483647.
2) Тип возвращаемого результата – BIGINT.
3) 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

1) Функция инициализирует датчик случайных чисел заданным <значением> или согласно текущему значению системного таймера, если <значение> не задано, меньше либо равно нулю или равно NULL.
2) Функция возвращает значение, которым был инициализирован датчик.
3) Тип возвращаемого результата – INT.

Определение имени текущего пользователя

Синтаксис

```
username()
```

Описание

Получение имени текущего пользователя.

Возвращаемое значение

Имя пользователя (char(66)), который вызвал процедуру.

Пример

```
1) Создание процедуры CurUser, владельцем которой является
 пользователь SYSTEM.
username SYSTEM/MANAGER8

create or replace procedure CurUser() result varchar(128)
code
  return "Текущий пользователь: " + trim(USERNAME());
end;

2) Выполнение процедуры.
execute CurUser();
Результат: Текущий пользователь: SYSTEM

3) Создание пользователя TESTER и предоставление ему прав на
 выполнение процедуры SYSTEM.CurUser().
create or replace user TESTER identified by '12345678';
grant execute on CurUser to TESTER;

4) Установка нового текущего пользователя соединения с СУБД.
username TESTER/12345678

5) Вызов на выполнение процедуры SYSTEM.CurUser пользователем
 TESTER.
execute SYSTEM.CurUser();
Результат: Текущий пользователь: TESTER
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
Примечание. Функция процедурного языка username() идентична
 псевдостолбцу USER языка SQL.
username SYSTEM/MANAGER8
create or replace procedure SelCurUser() result varchar(128)
declare
  var usr char(66);
code
  execute "select user;" into usr;
  return usr;
end;
execute SelCurUser();
Результат: SYSTEM
```

Определение идентификатора текущего пользователя

Синтаксис

```
userid()
```

Описание

Получение числового идентификатора текущего пользователя.

Возвращаемое значение

Значение типа `INT`, являющееся числовым идентификатором пользователя, от лица которого выполняется работа с СУБД ЛИНТЕР по текущему соединению.

Пример

```
create or replace procedure tst_userid() result int
code
  return userid(); //
end;
username SYSTEM/MANAGER8
execute tst_userid();
return value = 1
```

Определение имени фактического исполнителя процедуры

Синтаксис

```
effective_username()
```

Описание

Получение имени пользователя, с правами которого выполняется процедура.

В СУБД ЛИНТЕР с хранимыми процедурами связаны два понятия:

1) номинальный исполнитель процедуры;
2) фактический исполнитель процедуры.

Номинальный исполнитель – это текущий пользователь открытого соединения, который вызвал на исполнение процедуру. Этим пользователем может быть как владелец процедуры, так и любой пользователь, которому владелец процедуры предоставил право её выполнять.

Фактический исполнитель – пользователь БД, которому владелец процедуры предоставил право выполнять процедуру от своего имени. То есть в данном случае вызов процедуры с опцией `as owner` осуществляет текущий (номинальный) пользователь, а реально выполняет её другой пользователь. Функция `effective_username()` предоставляет имя того пользователя, с чьими правами будет реально выполняться процедура.

Возвращаемое значение

Имя пользователя (char(66)), с правами которого выполняется процедура.

Пример

```
1) Создание процедуры CurUser, владельцем которой является
 пользователь SYSTEM.
username SYSTEM/MANAGER8

create or replace procedure CurUser() result varchar(128)
code
  return "Номинальный: " + trim(USERNAME()) + ". " + "Фактический:
 " + trim(EFFECTIVE_USERNAME());
end;
2) Выполнение процедуры.
execute CurUser()
Результат: Номинальный: SYSTEM. Фактический: SYSTEM
3) Создание пользователя TESTER и предоставление ему владельцем
 процедуры CurUser прав на выполнение
этой процедуры от имени владельца.
create or replace user TESTER identified by '12345678';
grant execute as owner on CurUser to TESTER;
4) Выполнение процедуры от имени пользователя, которому дано это
 право.
В данном случае процедура запускается на выполнение текущим
 пользователем TESTER (номинальный исполнитель), а реально
 выполнятся с правами пользователя SYSTEM (фактический
 исполнитель)
username TESTER/12345678
execute SYSTEM.CurUser() as owner;
Результат: Номинальный: TESTER. Фактический: SYSTEM

Примечание. В случае если в процедуре используются таблицы
 БД, владельцами которых являются номинальный и фактический
 исполнители процедуры, то в имени таблицы должен указываться их
 владелец с помощью функции username() или effetive_username(),
 например,
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

"trim(username())".auto или "trim(effective _username())".auto.

Определение идентификатора фактического исполнителя процедуры

Синтаксис

```
effective_userid()
```

Описание

Функция предоставляет системный идентификатор фактического исполнителя процедуры (см. функцию effective_username()).

Возвращаемое значение

Значение типа INT, являющееся числовым идентификатором пользователя, с чьими правами выполняется процедура.

Пример

```
username SYSTEM/MANAGER8
create or replace procedure tst_userid() result int
code
  return effective_userid(); //
end;
execute tst_userid();
return value = 1
```

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

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

Пример

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

create or replace 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(?,?);" using uname, encstr; //
end;

create or replace 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 "select * from upwds where uname= ? and pwd = ?;" using
 uname, encstr; //
  if errcode() = 2 then
    return 0; // not correct
  else
    return 1; //
  endif
end;
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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, то есть 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.

Примеры

```
1)
create or replace procedure proc_re1(in i integer; out res
 integer) result integer for debug
declare
  exception DIVZERO for DIVZERO; //
  var ch int; //
code
  res:=50; //
  ch:=100/i; //
  res:=res/2; //
  return ch; //
exceptions
  when DIVZERO then
    raise_error (10333); //
end;
call proc_re1(0);
[Linter Code 10333] Native error 10333

2)
create or replace procedure proc_re1(in i integer; out res
 integer) for debug
declare
  var ch int; //
code
  res:=20*i; //
  if res < 100 then
    raise_error (10555); //
  endif; //
```

```
end;

3)
execute block result int
declare
  var S int; //
  var I int; //
  var c cursor(i int); //
code
  open c for "select max(personid) from auto;"; //
  S := c.i + 1; //
  close c; //
  I := 0; //
  while (I < S) loop
    I := I + 1; //
    raise_error(10299); //
  endloop; //
  return 0;  //
exceptions
  when all then
    resignal; // Error to up
end;
[Linter Code 10299] Native error 10299

4)
execute block result int
declare
  var S int; //
  var I int; //
  var c cursor(i int); //
code
  open c for "select max(personid) from auto;"; //
  S := c.i + 1; //
  close c; //
  I := S/0; //
  return S; //
exceptions
  when all then
    raise_error(10299); // Error to up
end;
[Linter Code 10299] Native error 10299
```

Момент срабатывания триггера

Синтаксис

```
sysevent()
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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

1) При использовании в хранимой процедуре всегда возвращается 0.
2) Целочисленное значение типа INT, являющееся комбинацией масок возможных событий (определение масок событий содержится в файле global.h). Свойства масок событий приведены в таблице 10.

Таблица 10. Свойства масок событий

Мнемоническое обозначение масок событий	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 "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` (константа).

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.

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

Примечание
На определенных платформах по объективным причинам разрешающая способность таймера может быть грубее ожидаемой.

Примеры

```
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:
- выполнить 20 циклов по 1 секунде:
execute test_sleep1(20);
- выполнить 4 цикла по 5 секунд:
execute test_sleep2(4);
Из протокола выполнения видно, что процедуры действительно
 «засыпают» и «просыпаются» через заданный промежуток времени.
2) Команда sleep полностью приравнена к select-запросу. Например,
 select-запрос,
 возвращающий 4 записи, будет 4 раза вызывать процедуру (с 5
 секундной задержкой
```

```
 внутри), общее время выполнения запроса будет порядка 20 секунд,
 параллельно
 работающие запросы будут выполняться без задержек:
create or replace procedure killer2(in j int) result int
declare
  var i int; //
code
  sleep(5000); //
  i := j+100; //
  return i; //
end;

!4 rows, 4x5=20 sec:
select sysdate;
select killer2(personid), make from auto where personid < 5;
select sysdate;
```

Одновариантная замена NULL-значения реальным значением

Синтаксис

```
nvl(<выражение 1>, <выражение 2>)

     <выражение 1>::=<значимое выражение>

     <выражение 2>::=<значимое выражение>
```

Описание

Функция возвращает значение <выражения 1> в случае, если <выражение 1> не является NULL-значением, иначе возвращает значение <выражения 2>.

Функция эквивалентна конструкции СУБД ЛИНТЕР:

```
CASE WHEN <выражение 1> IS NOT NULL
THEN <выражение 1> ELSE <выражение 2> END;
```

Типы данных <выражения 1> и <выражения 2> должны быть идентичными или приводимыми. При приведении типов данных по умолчанию (без конструкции CAST...) второй аргумент приводится к типу данных первого аргумента.

Пример

```
create procedure nvl_test(in arg1 char(20); in arg2 char(10))
 result char(20)
code
  return nvl(arg1, arg2);
end;

execute nvl_test("abc",null);
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

```
return value = abc

execute nvl_test(null,"123");
return value = 123
```

Двухвариантная замена NULL-значения реальным значением

Назначение

Двухвариантная замена неопределенного значения (NULL-значения) реальным значением.

Синтаксис

```
nvl2(<выражение 1>, <выражение 2>, <выражение 3>)

     <выражение 1>::=<значимое выражение>

     <выражение 2>::=<значимое выражение>

     <выражение 3>::=<значимое выражение>
```

Описание

Функция возвращает значение <выражения 2> в случае, если <выражение 1> не является NULL-значением, иначе возвращает значение <выражения 3>.

Функция эквивалентна конструкции СУБД ЛИНТЕР:

```
CASE WHEN <выражение 1> IS NOT NULL
THEN <выражение 2> ELSE <выражение 3> END;
```

Типы данных <выражения 1>, <выражения 2>, <выражения 3> должны быть идентичными или приводимыми. При приведении типов данных по умолчанию (без конструкции CAST…) второй и третий аргументы приводятся к типу данных первого аргумента.

Пример

```
create or replace procedure nvl_test(in v char(20)) result
 char(20)
code
  return nvl2(v, "is NOT NULL", "is NULL");
end;

execute nvl_test();
return value = is NULL

execute nvl_test('A');
return value = is NOT NULL
```

Вывод сообщения

Синтаксис

```
print(<строка>)
```

<строка> – текст сообщения

Описание

Выводит на консоль ядра СУБД ЛИНТЕР сообщение вида

```
*** Message from Stored Procedure: <текст сообщения>
```

если ядро запущено с ключом /PROCPRINT.

<Текст сообщения> должен быть англоязычным.

Максимальный размер выводимого сообщения 980 символов.

Примечание
Функция должна использоваться только при отладке хранимой процедуры.

Возвращаемое значение

1) Длина выводимой строки, если при запуске ядра СУБД ЛИНТЕР был задан ключ /PROCPRINT, и 0 в противном случае.

Пример

```
create or replace procedure print_example(in str char(100)) result
 int
declare
  var i int; //
code
  i:=  print("str = '" + str + "'"); //
  return i; //
end;

call print_example("123");

Результат на консоли ядра СУБД:
*** Message from Stored Procedure: str = '123'.
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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. Если же в процедурном канале не был установлен транзакционный режим, то 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`), 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`».

Примечание
Режим `OPTIMISTIC` устарел (использовать не рекомендуется).

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 устанавливается режим `EXCLUSIVE`.

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;
```

 © Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

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 "insert into test values (1);"; //
  begin transaction; //
  execute "insert into test values (2);"; //
  rollback transaction; //
  execute "insert into test values (3);"; //
  commit transaction; //
end;
execute tr_test();
select * from test;

I
-
|          1|
|          3|
```

Приложение

Ключевые слова процедурного языка

ABSOLUTE	ALL	ALTER
AND	APPEND	ARRAY
AS	AUTHID	BEGIN
BIGINT	BLOB	BLOCK
BOOL	BREAK	BSON
BY	BYTE	CALL
CASE	CHAR	CLOSE
CODE	COLUMN	COMMIT
CONTINUE	CREATE	CURRENT_USER
CURSOR	DATE	DEBUG
DECLARE	DEFAULT	DEFINER
DIRECT	DOUBLE	EIF
ELSE	ELSEIF	END
ENDCASE	ENDIF	ENDLOOP
EXCEPTION	EXCEPTIONS	EXECUTE
FALSE	FETCH	FIRST
FOR	FROM	GOTO
IF	IGNORE	IN
INOUT	INT	INTEGER
INTO	LAST	LIKE
LOOP	NCHAR	NEXT
NOT	NULL	NUMERIC
NVARCHAR	OF	OPEN
OR	OTHERS	OUT
PREVIOUS	PROCEDURE	PUTM
REAL	RELATIVE	RELEASE
REPLACE	RESIGNAL	RESULT
RETURN	ROLLBACK	SIGNAL
SLEEP	SMALLINT	START
STRUCT	THEN	TRANSACTION
TRUE	TYPEOF	UNTIL
USING	VAR	VARBYTE
VARCHAR	WHEN	WHILE
WORK	XMLREF	XMLTYPE

Предметный указатель

<BLOB-тип>, 13
<UNICODE тип переменной длины>, 13
<UNICODE тип фиксированной длины>, 13
<UNICODE тип>, 13
<байтовый тип переменной длины>, 13
<байтовый тип фиксированной длины>, 13
<байтовый тип>, 13
<внешний файл>, 13
<выполнение временной процедуры>, 29
<выполнение хранимой процедуры>, 24
<дата-время тип>, 13
<идентификатор>, 8
<имя глобальной переменной>, 13
<имя параметра>, 13
<имя переменной>, 13
<имя пользователя>, 13
<имя представления>, 13
<имя столбца>, 13
<имя схемы>, 13
<имя таблицы>, 13
<имя хранимой процедуры>, 13
<комментарий>, 8
<компиляция хранимой процедуры>, 21
<лексема>, 8
<логический тип>, 13
<модификация хранимой процедуры>, 21
<определение привилегий на хранимую процедуру>, 23
<отмена привилегий на хранимую процедуру>, 24
<приближенный числовой тип>, 13
<разделитель>, 8
<создание хранимой процедуры>, 18
<стандартный идентификатор>, 8
<строковый тип переменной длины>, 13
<строковый тип фиксированной длины>, 13
<строковый тип>, 13
<тип данных>, 13
<точный числовой тип>, 13
<удаление текста хранимой процедуры>, 27
<удаление хранимой процедуры>, 27
<универсальный идентификатор>, 8
выражение, 48
запрос динамический, 58
запрос претранслируемый, 58
идентификатор, 13
имя, 13
инициализатор, 18
литерал DATE, 11
литерал UNICODE, 11
литерал логический, 12
литерал символьный, 10
литерал числовой, 10
модификатор, 18
процедурный блок, 35

A
ABSOLUTE, 56
AFTER, 32
AND, 66
APPENDACTIVE, 36
APPENDNOTSTARTED, 36
APPLICATIONERROR, 36
ARRAY, 35
AS, 31, 55
AS OWNER, 24
AUTHID, 18

B
BADCODE, 36
BADCURSOR, 36
BADINDEX, 36
BADPARAM, 36, 92, 93, 94, 95
BADRETVAL, 36
BEFORE, 32
BIGINT, 10, 13, 95
BOOL, 13, 95
BREAK, 53
BSON, 91
BY, 52
BYTE, 13, 95

C
CALL, 24, 55
CASE, 49
CHAR, 13
CLOSE, 58
CODE, 37
COMMIT, 62
CONTINUE, 53
CURNOTOPEN, 36, 93, 94, 95
CURRENT_USER, 18
CURSOR, 13
CUSTOM, 36

D
DATE, 13, 95
DECLARE, 35
DEFAULT, 18, 35
DEFINER, 18
DELETING, 32
DIRECT, 55, 58
DIVZERO, 36
DOUBLE, 13, 95

E
EIF, 70
ELSE, 48, 70
ELSEIF, 48
END, 35, 37
ENDIF, 48
ENDLOOP, 50, 52
ENSCASE, 49
ERRTYPOPERAND, 95
EXCEPTION, 35, 36
EXCEPTIONS, 41
EXECUTE, 24, 58

F
FALSE, 12
FETCH, 56, 93
FIRST, 56, 93
FOR, 36, 52, 55, 91, 92
FOR DEBUG, 18

G
GOTO, 53

I
IF, 48
IGNORE, 41
IN, 18
INOUT, 18
INSERTING, 32
INSTEAD OF, 32
INT, 13
INTEGER, 10, 13, 95
INTO, 24, 56, 58
INVTRSTATE, 36

L
LAST, 56
LOOP, 50, 51, 52

N
NCHAR, 13
NEW, 31
NEXT, 56, 93
NOMEM, 36
NOT, 66
NULL, 12
NULLDATA, 12, 36
NUMERIC, 13
NVARCHAR, 13

O
OLD, 31
OPEN, 55, 91, 92
OR, 66
OUT, 18
OVERFLOW, 36

P
PREVIOUS, 56
PROCEDURE, 18

Q
QUERYWHENAPPEND, 36

R
REAL, 13
RELATIVE, 56
RELEASE, 62
RESIGNAL, 37, 54
RESULT, 18, 18
RETURN, 54
ROLLBACK, 62
ROWCOUNT, 33

S
SESSIONID, 33
SIGNAL, 54
SMALLINT, 10, 13

T
THEN, 41, 48, 49
TRIGQUERY, 36
TRUE, 12
TYPEOF, 18

U
UNDEFPROC, 36
UNTIL, 51
UPDATING, 32
USING, 55, 58

V
VAR, 18, 35
VARBYTE, 13
VARCHAR, 13, 95

W
WHEN, 41, 49
WHEN ALL, 41
WHEN OTHERS, 41, 49
WHILE, 50, 52

Указатель операторов

выражение, 48

A
ALTER PROCEDURE, 21
ALTER PROCEDURE DROP TEXT, 27

B
BREAK, 53

C
CALL, 24
CASE ... WHEN ... THEN, 49
CLOSE, 58
COMMIT, 62
CONTINUE, 53
CREATE PROCEDURE, 18

D
DROP PROCEDURE, 27

E
end append, 71
EXECUTE, 24, 58
EXECUTE BLOCK, 29
EXECUTE DIRECT, 58
EXECUTE PROCEDURE, 24

F
FETCH, 56
FOR...WHILE...LOOP, 52

G
GOTO, 53
GRANT, 23

I
IF ... THEN ... ELSE, 48

L
LOOP ... UNTIL, 51

O
OPEN ... FOR, 55
OPEN ... FOR BSON, 91
OPEN ... FOR CALL, 55

P
putm, 71

R
REBUILD PROCEDURE, 21
RESIGNAL, 54
RETURN, 54
REVOKE, 24
ROLLBACK, 62

S
SIGNAL, 54
start append, 71

W
WHILE ... LOOP, 50

Указатель функций

&, 66
|, 67

A
abs, 123
acos, 126
add_blob, 78, 79
add_month, 134
asc, 112
ascii, 113
asin, 126
atan, 126
atan2, 126
atod, 141

B
begin transaction, 168
blob_size, 78, 85
btoa, 141

C
ceil, 123
char_length, 104
chr, 112
clear_blob, 78, 86
clearPutm, 71
commit transaction, 168
cos, 126
cosh, 127
currow, 149

D
date_round, 124
date_trunc, 125
datesplit, 135
day, 129
dbname, 157
difference, 117
divtime, 137
dtoa, 139
dupchar, 112

E
effective_userid, 157
effective_username, 155
encode_string, 157
errcode, 147
exccode, 148
exp, 127
extractbigint, 94
extractbool, 94
extractbytes, 94
extractdate, 94
extractdouble, 94
extractint, 94
extractstring, 94
extractvalue, 96

F
floor, 123
flushPutm, 71
ftoa, 138

G
getPutmRecs, 71
greatest, 150

H
hextoraw, 145
hour, 131

I
initcap, 114
insert, 107
instr, 103
int getPutmRecs, 71
itoa, 137

L
last_day, 133
least, 152
len, 104
length, 78, 104
ln, 127
localdate, 133
log, 127
lower, 113
lpad, 99
ltrim, 105

M
make_date, 132
makestr, 110
max, 149
min, 151
minute, 131
mod, 153
modify_blob_type, 78, 87
month, 130
multime, 136

N
next_day, 133
ninitcap, 121
ninsert, 122

© Процедурный язык. АО НПП «РЕЛЭКС», 1990-2026

nlen, 117
nlower, 120
nlpad, 118
nltrim, 121
noverlay, 122
nrepeat_string, 119
nreplace, 122
nright_substr, 119
nrpad, 119
nrtrim, 121
nstrpos, 120
nsubstr, 118
ntoa, 139
ntolower, 120
ntoupper, 120
ntranslate, 122
ntrim, 118
nupper, 120
nvl, 164
nvl2, 165

O
octet_length, 104
outofcursor, 146
overlay, 107

P
print, 166

R
raise_error, 159
rand, 153
randomize, 153
rawtohex, 145
read_blob, 78, 81
read_blob_bigint, 78, 82
read_blob_bool, 78, 85
read_blob_char, 78, 83
read_blob_date, 78, 84
read_blob_double, 78, 83
read_blob_int, 78, 81
read_blob_nchar, 78, 84
read_blob_numeric, 78, 83
read_blob_real, 78, 82
read_blob_smallint, 78, 81
repeat_string, 102
replace, 109
right_substr, 101
rollback transaction, 168
round, 128
rowcount, 146
rpad, 100
rtrim, 106

S
second, 131
seek_blob, 78, 85
set_cur_blob, 78, 79
sign, 129
sin, 126
sinh, 127
sleep, 162
soundex, 116
sqlerrm, 147
sqrt, 129
strpos, 107
substr, 101
sysdate, 132
sysevent, 160

T
tan, 126
tanh, 127
ticks, 132
to_char, 114
to_date, 141
tobigint, 144
tochar, 141
tointeger, 144
tolower, 113
tonchar, 120
tonumeric, 145
toreal, 144
tosmallint, 143
toupper, 113
translate, 110
trim, 105
trunc, 128

U
upper, 113
userid, 155
username, 154

Y
year, 130