**Тема 6. Язык запросов SQL. Язык запросов SQL. Пользовательские функции и процедуры.**

**PL/pgSQL — процедурный язык SQL**

PL/pgSQL это процедурный язык для СУБД Postgres Pro. Целью проектирования PL/pgSQL было создание загружаемого процедурного языка, который:

* используется для создания функций и триггеров,
* добавляет управляющие структуры к языку SQL,
* может выполнять сложные вычисления,
* наследует все пользовательские типы, функции и операторы,
* может быть определён как доверенный язык,
* прост в использовании.

Функции PL/pgSQL могут использоваться везде, где допустимы встроенные функции. Например, можно создать функции со сложными вычислениями и условной логикой, а затем использовать их при определении операторов или в индексных выражениях.

Функции, написанные на PL/pgSQL, определяются на сервере командами CREATE FUNCTION. Такая команда обычно выглядит, например, так:

CREATE FUNCTION somefunc(integer, text) RETURNS integer

AS 'тело функции'

LANGUAGE plpgsql;

Тело функции представляет собой просто текстовую строку. Часто для написания тела функции удобнее заключать эту строку в знаки долларов, а не в обычные апострофы. Если не применять заключение в доллары, все апострофы или обратные косые черты в теле функции придется экранировать.

PL/pgSQL это блочно-структурированный язык. Текст тела функции должен быть блоком. Структура блока:

[ <<метка>> ]

[ DECLARE

объявления ]

BEGIN

операторы

END [ метка ];

Каждое объявление и каждый оператор в блоке должны завершаться символом ";"(точка с запятой). Блок, вложенный в другой блок, должен иметь точку с запятой после END, однако финальный END, завершающий тело функции, не требует точки с запятой.

Метка требуется только тогда, когда нужно идентифицировать блок в операторе EXIT, или дополнить имена переменных, объявленных в этом блоке. Если метка указана после END, то она должна совпадать с меткой в начале блока.

Ключевые слова не чувствительны к регистру символов. Как и в обычных SQL-командах, идентификаторы неявно преобразуются к нижнему регистру, если они не взяты в двойные кавычки.

Комментарии в PL/pgSQL коде работают так же, как и в обычном SQL. Двойное тире (--) начинает комментарий, который завершается в конце строки. Блочный комментарий начинается с /\* и завершается \*/. Блочные комментарии могут быть вложенными.

Любой оператор в выполняемой секции блока может быть вложенным блоком. Вложенные блоки используются для логической группировки нескольких операторов или локализации области действия переменных для группы операторов. Во время выполнения вложенного блока переменные, объявленные в нём, скрывают переменные внешних блоков с такими же именами. Чтобы получить доступ к внешним переменным, нужно дополнить их имена меткой блока. Например:

CREATE FUNCTION somefunc() RETURNS integer AS $$

<< outerblock >>

DECLARE

quantity integer := 30;

BEGIN

RAISE NOTICE 'Сейчас quantity = %', quantity; -- Выводится 30

quantity := 50;

--

-- Вложенный блок

--

DECLARE

quantity integer := 80;

BEGIN

RAISE NOTICE 'Сейчас quantity = %', quantity; -- Выводится 80

RAISE NOTICE 'Во внешнем блоке quantity = %', outerblock.quantity; -- Выводится 50

END;

RAISE NOTICE 'Сейчас quantity = %', quantity; -- Выводится 50

RETURN quantity;

END;

$$ LANGUAGE plpgsql;

Все переменные, используемые в блоке, должны быть определены в секции объявления. (За исключением переменной-счетчика цикла FOR, которая объявляется автоматически. Для цикла по диапазону чисел автоматически объявляется целочисленная переменная, а для цикла по результатам курсора - переменная типа record.)

Переменные PL/pgSQL могут иметь любой тип данных SQL, такой как integer, varchar, char.

Примеры объявления переменных:

user\_id integer;

quantity numeric(5);

url varchar;

Предложение DEFAULT, если присутствует, задает начальное значение, которое присваивается переменной при входе в блок. Если отсутствует, то переменная инициализируется SQL-значением NULL. Указание CONSTANT предотвращает изменение значения переменной после инициализации, таким образом, значение остается постоянным в течение всего блока. Параметр COLLATE определяет правило сортировки, которое будет использоваться для этой переменной. Если указано NOT NULL, то попытка присвоить NULL во время выполнения приведёт к ошибке. Все переменные, объявленные как NOT NULL, должны иметь непустые значения по умолчанию. Для присваивания можно использовать знак равенства (=) вместо совместимого с PL/SQL :=.

Переданные в функцию параметры именуются идентификаторами $1, $2 и т. д. Дополнительно, для улучшения читаемости, можно объявить псевдонимы для параметров $n. Либо псевдоним, либо цифровой идентификатор используются для обозначения параметра. Предпочтительный способ создать псевдоним - это дать имя параметру в команде CREATE FUNCTION, например:  
  
CREATE FUNCTION sales\_tax(subtotal real) RETURNS real AS $$

BEGIN

RETURN subtotal \* 0.06;

END;

$$ LANGUAGE plpgsql;

Другой способ это явное объявление псевдонима при помощи синтаксиса ALIAS:

CREATE FUNCTION sales\_tax(real) RETURNS real AS $$

DECLARE

subtotal ALIAS FOR $1;

BEGIN

RETURN subtotal \* 0.06;

END;

$$ LANGUAGE plpgsql;

**Основные операторы PL/pgSQL. Выполнение SQL-скриптов внутри PL/pgSQL**

В функции на PL/pgSQL можно выполнить любую команду SQL, не возвращающую строк, просто написав эту команду (например, INSERT без предложения RETURNING):

CREATE FUNCTION somefunc() RETURNS integer AS $$

<< outerblock >>

DECLARE

quantity integer := 30;

BEGIN

RAISE NOTICE 'Сейчас quantity = %', quantity; -- Выводится 30

quantity := 50;

--

-- Вложенный блок

--

DECLARE

quantity integer := 80;

BEGIN

RAISE NOTICE 'Сейчас quantity = %', quantity; -- Выводится 80

RAISE NOTICE 'Во внешнем блоке quantity = %', outerblock.quantity; -- Выводится 50

INSERT INTO t1 VALUES (3 \* quantity, 'Hi! I am from function somefunc');

END;

RAISE NOTICE 'Сейчас quantity = %', quantity; -- Выводится 50

RETURN quantity;

END;

Результат SQL-команды, возвращающей одну строку (возможно из нескольких столбцов), может быть присвоен переменной типа record, переменной строкового типа или списку скалярных переменных. Для этого нужно к основной команде SQL добавить предложение INTO. Так, например:

CREATE FUNCTION somefunc() RETURNS integer AS $$

<< outerblock >>

DECLARE

quantity integer := 30;

countofrecords integer;

BEGIN

RAISE NOTICE 'Сейчас quantity = %', quantity; -- Выводится 30

quantity := 50;

SELECT count(\*) INTO countofrecords FROM t1;

RAISE NOTICE 'Сейчас countofrecords = % %', countofrecords, quantity;

--

-- Вложенный блок

--

DECLARE

quantity integer := 80;

BEGIN

RAISE NOTICE 'Сейчас quantity = %', quantity; -- Выводится 80

RAISE NOTICE 'Во внешнем блоке quantity = %', outerblock.quantity; -- Выводится 50

INSERT INTO t1 VALUES (3 \* quantity, 'Hi! I am from function somefunc');

END;

RAISE NOTICE 'Сейчас quantity = %', quantity; -- Выводится 50

RETURN quantity;

END;

$$ LANGUAGE plpgsql;

SELECT somefunc();

**Условные операторы**

Операторы IF и CASE позволяют выполнять команды в зависимости от определённых условий. PL/pgSQL поддерживает три формы IF:

IF ... THEN ... END IF

IF ... THEN ... ELSE ... END IF

IF ... THEN ... ELSIF ... THEN ... ELSE ... END IF

и две формы CASE:

CASE ... WHEN ... THEN ... ELSE ... END CASE

CASE WHEN ... THEN ... ELSE ... END CASE

Пример:

CREATE OR REPLACE FUNCTION somefunc() RETURNS integer AS $$

<< outerblock >>

DECLARE

quantity integer := 30;

countofrecords integer;

msg varchar;

BEGIN

RAISE NOTICE 'Сейчас quantity = %', quantity; -- Выводится 30

quantity := 50;

if quantity > 49 THEN

SELECT count(\*) INTO countofrecords FROM t1;

ELSIF quantity > 50 THEN

SELECT count(\*) INTO countofrecords FROM t2;

END IF;

CASE

WHEN countofrecords BETWEEN 8 AND 10 THEN

msg := 'значение в диапазоне между 0 и 10';

WHEN countofrecords BETWEEN 11 AND 20 THEN

msg := 'значение в диапазоне между 11 и 20';

END CASE;

RAISE NOTICE 'msg = %', msg;

RAISE NOTICE 'Сейчас countofrecords = % %', countofrecords, quantity;

--

-- Вложенный блок

--

DECLARE

quantity integer := 80;

BEGIN

RAISE NOTICE 'Сейчас quantity = %', quantity; -- Выводится 80

RAISE NOTICE 'Во внешнем блоке quantity = %', outerblock.quantity; -- Выводится 50

INSERT INTO t1 VALUES (3 \* quantity, 'Hi! I am from function somefunc');

END;

RAISE NOTICE 'Сейчас quantity = %', quantity; -- Выводится 50

RETURN quantity;

END;

$$ LANGUAGE plpgsql;

**Циклы**

Операторы LOOP, EXIT, CONTINUE, WHILE, FOR и FOREACH позволяют повторить серию команд в функции на PL/pgSQL:

[<<метка>>]

LOOP

операторы

END LOOP [ метка ];

LOOP организует безусловный цикл, который повторяется до бесконечности, пока не будет прекращен операторами EXIT или RETURN. Для вложенных циклов можно использовать метку в операторах EXIT и CONTINUE, чтобы указать, к какому циклу эти операторы относятся.

**Циклы (EXIT)**

EXIT [ метка ] [WHEN логическое-выражение];

Если метка не указана, то завершается самый внутренний цикл, далее выполняется оператор, следующий за END LOOP. Если метка указана, то она должна относиться к текущему или внешнему циклу, или это может быть метка блока. При этом в именованном цикле/блоке выполнение прекращается, а управление переходит к следующему оператору после соответствующего END. При наличии WHEN цикл прекращается, только если логическое-выражение истинно. В противном случае управление переходит к оператору, следующему за EXIT. EXIT можно использовать со всеми типами циклов, не только с безусловным. Когда EXIT используется для выхода из блока, управление переходит к следующему оператору после окончания блока. Обратите внимание, что для выхода из блока нужно обязательно указывать метку. EXIT без метки не позволяет прекратить работу блока

**Циклы (CONTINUE)**

CONTINUE [ метка ] [WHEN логическое-выражение];

Если метка не указана, то начинается следующая итерация самого внутреннего цикла. Если метка присутствует, то она указывает на метку цикла, выполнение которого будет продолжено. При наличии WHEN следующая итерация цикла начинается только тогда, когда логическое-выражение истинно. В противном случае управление переходит к оператору, следующему за CONTINUE. CONTINUE можно использовать со всеми типами циклов, не только с безусловным.

Примеры:

LOOP

-- здесь производятся вычисления

EXIT WHEN count > 100;

CONTINUE WHEN count < 50;

-- вычисления для count в диапазоне 50 .. 100

END LOOP;

**Циклы (WHILE)**

WHILE выполняет серию команд до тех пор, пока истинно логическое-выражение. Выражение проверяется непосредственно перед каждым входом в тело цикла.

[<<метка>>]

WHILE логическое-выражение LOOP

операторы

END LOOP [ метка ];

**Циклы (Целочисленный вариант FOR)**

[<<метка>>]

FOR имя IN [REVERSE] выражение .. выражение [BY выражение] LOOP

операторы

END LOOP [ метка ];

В этой форме цикла FOR итерации выполняются по диапазону целых чисел. Переменная имя автоматически определяется с типом integer и существует только внутри цикла (если уже существует переменная с таким именем, то внутри цикла она будет игнорироваться). Выражения для нижней и верхней границы диапазона чисел вычисляются один раз при входе в цикл. Если не указано BY, то шаг итерации 1, в противном случае используется значение в BY, которое вычисляется, опять же, один раз при входе в цикл. Если указано REVERSE, то после каждой итерации величина шага вычитается, а не добавляется.

Примеры целочисленного FOR:

FOR i IN 1..10 LOOP

-- внутри цикла переменная i будет иметь значения 1,2,3,4,5,6,7,8,9,10

END LOOP;

FOR i IN REVERSE 10..1 LOOP

-- внутри цикла переменная i будет иметь значения 10,9,8,7,6,5,4,3,2,1

END LOOP;

FOR i IN REVERSE 10..1 BY 2 LOOP

-- внутри цикла переменная i будет иметь значения 10,8,6,4,2

END LOOP;

Если нижняя граница цикла больше верхней границы (или меньше, в случае REVERSE), то тело цикла не выполняется вообще. При этом ошибка не возникает.

Если с циклом FOR связана метка, к целочисленной переменной цикла можно обращаться по имени, указывая эту метку.

**Циклы (Цикл по результатам запроса)**

Другой вариант FOR позволяет организовать цикл по результатам запроса. Синтаксис:

[ <<метка>> ]

FOR цель IN запрос LOOP

операторы

END LOOP [ метка ];

Переменная цель может быть строковой переменной, переменной типа record или разделённым запятыми списком скалярных переменных. Переменной цель последовательно присваиваются строки результата запроса, и для каждой строки выполняется тело цикла. Пример:

CREATE OR REPLACE FUNCTION somefunc() RETURNS integer AS $$

<< outerblock >>

DECLARE

quantity integer := 30;

i integer := 0;

countofrecords integer;

msg varchar;

tr RECORD;

BEGIN

RAISE NOTICE 'Сейчас quantity = %', quantity; -- Выводится 30

quantity := 50;

if quantity > 49 THEN

SELECT count(\*) INTO countofrecords FROM t1;

ELSIF quantity > 50 THEN

SELECT count(\*) INTO countofrecords FROM t2;

END IF;

LOOP

RAISE NOTICE 'i = %', i; -- Выводится 30

i := i + 1;

EXIT WHEN i >= 50;

END LOOP;

WHILE i < 100 LOOP

RAISE NOTICE 'i = %', i; -- Выводится 30

i := i + 1;

END LOOP;

FOR tr IN

SELECT \* FROM t1

LOOP

RAISE NOTICE 'result num % name %', tr.num, tr.name;

END LOOP;

CASE

WHEN countofrecords BETWEEN 8 AND 10 THEN

msg := 'значение в диапазоне между 0 и 10';

WHEN countofrecords BETWEEN 11 AND 20 THEN

msg := 'значение в диапазоне между 11 и 20';

END CASE;

RAISE NOTICE 'msg = %', msg;

RAISE NOTICE 'Сейчас countofrecords = % %', countofrecords, quantity;

--

-- Вложенный блок

--

DECLARE

quantity integer := 80;

BEGIN

RAISE NOTICE 'Сейчас quantity = %', quantity; -- Выводится 80

RAISE NOTICE 'Во внешнем блоке quantity = %', outerblock.quantity; -- Выводится 50

INSERT INTO t1 VALUES (3 \* quantity, 'Hi! I am from function somefunc');

END;

RAISE NOTICE 'Сейчас quantity = %', quantity; -- Выводится 50

RETURN quantity;

END;

$$ LANGUAGE plpgsql;

В качестве запроса в этом типе оператора FOR может задаваться любая команда SQL, возвращающая строки. Чаще всего это SELECT, но также можно использовать и INSERT, UPDATE или DELETE с предложением RETURNING. Кроме того, возможно применение и некоторых служебных команд, например EXPLAIN.

Применение описанных механизмов возможно для решения конкретных прикладных задач с максимальной скоростью - например, для переливки таблиц:

CREATE OR REPLACE FUNCTION reloadT1Table(tablename varchar) RETURNS void AS $$

<< outerblock >>

DECLARE

tr RECORD;

BEGIN

EXECUTE format('CREATE TABLE %I (

num integer,

name varchar

)', tablename);

RAISE NOTICE 'Таблица % была создана', tablename;

FOR tr IN

SELECT \* FROM t1

LOOP

RAISE NOTICE 'Reload data num % name %', tr.num, tr.name;

EXECUTE format('INSERT INTO %I VALUES(%s, %L)', tablename, tr.num, tr.name);

END LOOP;

RAISE NOTICE 'Таблица t1 была перелита в % ', tablename;

RETURN;

END;

$$ LANGUAGE plpgsql;

SELECT reloadT1Table('newTable');

SELECT \* FROM public."newTable";