1. Что такое агрегатные функции?

Агрегатные функции получают единственный результат из набора входных значений.

Агрегатные функции array\_agg, json\_agg, json\_object\_agg, string\_agg

и xmlagg так же, как и подобные пользовательские агрегатные функции, выдают разные по смыслу результаты в зависимости от порядка входных данных. По умолчанию порядок не определён, но его можно задать, дополнив вызов агрегатной функции предложением ORDER BY.

1. Как выполнить пользовательскую функцию?

* **CREATE FUNCTION** *имя\_функции* ([[ *метод\_аргумента* ] [*имя\_аргумента* ] *тип\_ аргумента* [,…] ]) - после ключевых слов CREATE FUNCTION указывается имя создаваемой функции, после чего в круглых скобках перечисляются аргументы, разделенные запятыми. Для каждого аргумента достаточно указать только тип, но при желании можно задать метод (in, out, inout; по умолчанию in) и  имя.  Если список в круглых скобках пуст, функция вызывается без аргументов (хотя сами круглые скобки обязательно должны присутствовать как в определении функции, так и при ее использовании). Ключевые слова  OR REPLACE  используются для изменения уже существующей функции.
* **RETURNS** тип\_возвращаемого\_значения   -  тип данных, возвращаемый функцией.
* **AS** '*определение*'  -  программное определение функции. В процедурных языках (таких, как PL/pgSQL) оно состоит из кода функции. Для откомпилированных функций С указывается абсолютный системный путь к файлу, содержащему объектный код.
* **LANGUAGE** '*язык*'. Название языка, на котором написана функция. В аргументе может передаваться имя любого процедурного языка (такого, как plpgsql или plperl, если соответствующая поддержка была установлена при компиляции),  С или SQL.   
  **[**
* **WITH** ( *атрибут* [. ...] ) ] - *атрибут* может принимать два значения: iscachable  и  isstrict.  
        **iscachable**. Оптимизатор может использовать предыдущие вызовы функций для ускоренной обработки будущих вызовов с тем же набором аргументов. Кэширование обычно применяется при работе с функциями, сопряженны­ми с большими затратами ресурсов, но возвращающими один и тот же ре­зультат при одинаковых значениях аргументов.  
         **isstrict**. Функция всегда возвращает NULL в случае, если хотя бы один из ее аргументов равен NULL. При передаче атрибута isstrict результат возвраща­ется сразу, без фактического выполнения функции.

В PostgreSQL cоздание  функций на языке С разрешено только суперпользователям, поскольку эти функции могут содержать системные вызовы,  представляющие потенциальную угрозу для безопасности системы.

1. Синтаксис команды создания функции.

**CREATE**[ OR REPLACE ] **FUNCTION**  
имя\_функции**([ [** метод\_аргумента **][**имя\_аргумента **]** тип\_ аргумента **[,…] ])**  
**RETURNS**тип\_возвращаемого\_значения   
**AS  '**определение**'   
LANGUAGE  '**язык**'  
 [ WITH (**атрибут**[…])]**

1. Как в теле функции связать переменную со значением входного параметра?

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

1. Синтаксис языка PL/pgSQL.

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;

myrow tablename%ROWTYPE;

myfield tablename.columnname%TYPE;

arow RECORD;

Общий синтаксис объявления переменной:

***имя*** [ CONSTANT ] ***тип*** [ COLLATE ***имя\_правила\_сортировки*** ] [ NOT NULL ] [ { DEFAULT | := | = } ***выражение*** ];

Синтаксис функции на языке PL/pgSQL следующий:

*create [or replace] function <имя функции>(<аргументы>)*

*returns <тип возврата> as ‘<тело функции>’*

*language ‘plpgsql’;*

1. Возможно ли использование в теле функции запросов?

Да, при этом нужно использовать аргументы.

1. Как организовать условие в теле функции?

Пример:

CREATE FUNCTION new\_emp() RETURNS emp AS $$

SELECT text 'None' AS name,

1000.0 AS salary,

25 AS age,

point '(2,2)' AS cubicle;

$$ LANGUAGE SQL;

1. Как создать цикл в теле функции?

RETURN ***выражение***;

RETURN с последующим выражением прекращает выполнение функции и возвращает значение выражения в вызывающую программу. Эта форма используется для функций PL/pgSQL, которые не возвращают набор строк.

В функции, возвращающей скалярный тип, результирующее выражение автоматически приводится к типу возвращаемого значения. Однако, чтобы вернуть составной тип (строку), возвращаемое выражение должно в точности содержать требуемый набор столбцов. При этом может потребоваться явное приведение типов.

Для функции с выходными параметрами просто используйте RETURN без выражения. Будут возвращены текущие значения выходных параметров.

Для функции, возвращающей void, RETURN можно использовать в любом месте, но без выражения после RETURN.

Возвращаемое значение функции не может остаться не определённым. Если достигнут конец блока верхнего уровня, а оператор RETURN так и не встретился, происходит ошибка времени исполнения. Это не касается функций с выходными параметрами и функций, возвращающих void. Для них оператор RETURN выполняется автоматически по окончании блока верхнего уровня.

Несколько примеров:

-- Функции, возвращающие скалярный тип данных

RETURN 1 + 2;

RETURN scalar\_var;

-- Функции, возвращающие составной тип данных

RETURN composite\_type\_var;

RETURN (1, 2, 'three'::text); -- требуется приведение типов

Оператор цикла *FOR*. Наиболее распространенный цикл. Переменная цикла *<var>* - то самое исключение, когда переменная может не описываться в секции *DECLARE*. В этом случае зона ее видимости ограничивается циклом. Синтаксис:

*For <var> in <start>..<stop> loop*

*<тело цикла>*

*End loop;*

Существует вариант этого цикла и для «прохода» по результату выборки:

*For <row-var> in <select> loop*

*<тело цикла>*

*End loop;*

В этом случае в переменную *<row-var>* последовательно подставляются строки из выборки, и тип этой переменной должен быть либо *<table>%rowtype*, отражающий запись конкретной таблицы *<table>*, либо *RECORD*, описывающий обобщенную запись таблицы.

1. Используются ли ограничения, наложенные на таблицу, при добавлении в нее записей через созданную функцию?

**ограничения** CHECK проверяются только **при добавлении** или изменении строк, а не когда в выражении CHECK **используется** пользовательская **функция**, в **ней ограничение** внешнего ключа, ссылающееся на **таблицу** :

1. Можно ли использовать созданные функции за пределами базы данных, в которой они созданы? Почему?

Нет нельзя, т.к. функция созданная в пределах баз данных работает локально.