Разные полезные средства PostgreSQL...

Версия 1.01 от 21.12.2016

Ссылки

- http://postgresql.ru.net/
- https://postgrespro.ru/
- https://www.postgresql.org/
- https://wiki.postgresql.org/
- https://wiki.postgresql.org/wiki/%D0%A7%D0%B0%D1%81%D1%82%D0%BE_%D0%97%D0%B0%D0%B4%D0%B0%D0%B2%D0%B0%D0%B5%D0%BC%D1%8B%D0%B5_%D0%92%D0%BE%D0%BF%D1%80%D0%BE%D1%81%D1%8B

Функции

- Функции общего назначения
- Триггерные функции
- Специальные

Функции общего назначения

- Строки
- Числа
- UUID
- Логические
- Отладка (debug)

Строки

Функции:
- str_is_null
- str_build_left
- str_build_right
- str_build
- uuid_to_str
- str_words_count
- str_word_by_num
- param_val_from_str
- param_val_replace

str_is_null

-- Проверка строки (text) на null. Если null, то значение меняется на sdef

```
CREATE OR REPLACE FUNCTION str_is_null(
      sx text,
      sdef text DEFAULT ''::text)
  RETURNS text AS
$$
BEGIN
  IF (sx is NULL) THEN
       IF (sdef is NULL) THEN sdef=''; END IF;
       RETURN sdef;
  ELSE
    RETURN sx;
  END IF;
END;
$$
  LANGUAGE plpgsql;
```

str_build_left

-- Нарастить входную строку заданными символами до заданного размера слева

```
CREATE OR REPLACE FUNCTION str_build_left (
      sx text,
      newsize integer,
      ch char(1) DEFAULT '0')
RETURNS text AS
$$
DECLARE
  res text;
BEGIN
```

```
  /*
    Нарастить входную строку заданными символами до заданного размера слева
    sx - входня строка
    newsize - новый размер строки
    ch - символы, которыми "наращивается строка"
    Пример вызова:
        select str_build_left ('3',5,'0');
        Результат:   '00003'
  */

  sx = str_is_null(sx);
  res = sx;
  newsize = int_is_null(newsize);
  IF (newsize > 0) THEN
      IF (ch IS NULL) then ch =' '; END IF;
          WHILE (CHAR_LENGTH(res) < newsize) LOOP
            res = ch || res;
          END LOOP;
  END IF;
  res = str_is_null(res);

  RETURN res;

END;
$$ LANGUAGE plpgsql;
```

str_build_right

```
--      Нарастить входную строку заданными символами до заданного размера справа

CREATE OR REPLACE FUNCTION str_build_right (
     sx text,
     newsize integer,
     ch char(1) DEFAULT '0')
RETURNS text AS
$$
DECLARE
  res text;
BEGIN
  /*
    Нарастить входную строку заданными символами до заданного размера справа
    sx - входня строка
    newsize - новый размер строки
    ch - символы, которыми "наращивается строка"
    Пример вызова:
        select str_build_right ('3',5,'0');
        Результат:   '30000'
  */

  sx = str_is_null(sx);
```

```
  res = sx;
  newsize = int_is_null(newsize);
  IF (newsize > 0) THEN
      IF (ch IS NULL) then ch =' '; END IF;
          WHILE (CHAR_LENGTH(res) < newsize) LOOP
            res =   res || ch;
          END LOOP;
  END IF;
  res = str_is_null(res);

  RETURN res;

END;
$$ LANGUAGE plpgsql;
```

str_build

```
--      Нарастить входную строку заданными символами до заданного размера справа или слева


CREATE OR REPLACE FUNCTION str_build (
     sx text,
     newsize integer,
     ch char(1) DEFAULT '0',
     nr_where integer DEFAULT 0
     )
RETURNS text AS
$$
DECLARE
  res text;
  nl integer;
  nr integer;
BEGIN
  /*

     ДОРАБОТАТЬ для nr_where = 0

     Нарастить входную строку заданными символами до заданного размера справа или слева
     sx - входная строка
     newsize - новый размер строки
     ch - символы, которыми "наращивается строка"
     nr_where - флаг. Если <0, то слева. Если >0, то справа. Если =0, то слева и справа
(центрирование) . Для =0 пока не корректно
     Пример вызова:
         select str_build ('3',5,'0',-1);
         select str_build ('3',5,'0',1);
         select str_build ('3',5,'0',0);
         select str_build ('3',6,'0',0);
         select str_build ('3',7,'0',0);
         select str_build ('3',8,'0',0);
  */
```

```
  sx = str_is_null(sx);
  res = sx;
  newsize = int_is_null(newsize);

  -- PERFORM debuglog_clear();
  --PERFORM debuglog_add('str','str_build', 'sx', sx, '');
  --PERFORM debuglog_add('str','str_build', 'newsize', '' || newsize, '');

  IF (newsize > 0) THEN
     nr_where = int_is_null(nr_where);

        --PERFORM debuglog_add('str','str_build', 'nr_where', '' || nr_where, '');

     IF (ch IS NULL) then ch =' '; END IF;
     IF (nr_where < 0) THEN res = str_build_left(sx, newsize, ch); END IF;
     IF (nr_where > 0) THEN res = str_build_right(sx, newsize, ch); END IF;
     IF (nr_where = 0) THEN
          nl = div(newsize,2);
          nr = newsize - nl;
          res = str_build_left(sx, nr, ch);
          res = str_build_right(res, newsize, ch);
     END IF;
  END IF;
  res = str_is_null(res);

  RETURN res;

END;
$$ LANGUAGE plpgsql;
```

str_words_count

```
-- Вычислить кол-во слов в строке Sx.

-- select str_words_count('   ,,,1, ;;; 2, 3', ' ,');   -- 4 слова
-- select * from debuglog_view('Str','str_words_count');

CREATE OR REPLACE FUNCTION str_words_count(Sx text, sUnChar text DEFAULT ' ',
YesTrimBefore boolean DEFAULT true, sXrenovina text DEFAULT '|') RETURNS integer AS $$
DECLARE
     i integer;
     k integer;
     m integer;
     inword boolean;
     ch char(1);
     Res integer;
BEGIN
/*
2016.12.18     Вычислить кол-во слов в строке Sx.
Разделителями слов могут быть любые символы из sUnChar
```

Параметры:
  Sx - входная строка;
  sUnChar - НеСимволы. Строка (типа - множество НеСимволов);
  YesTrimBefore - если TRUE, то перед обработкой удаляются НеСимволы слева и справа в Sx
  sXrenovina - подстрока, на которую заменяются пробелы (следствие НеЯсности для меня работы функций substring, position). Такого символа в принципе не должно быть в строке

Возвращаемое значение: кол-во слов (integer)

Примеры:
```
  select str_words_count('   ,,,1, ;;; 2, 3', ' ,');   -- 4 слова
  select * from str_words_count('1,2 ;3', ';, ');   -- 3 слова
  select * from str_words_count('1,2 ;3', ','); -- 2 слова
  select * from str_words_count('1,2 ;3', ';3' ); -- 1 слово
  select * from str_words_count('1,2      ;3', ' ' ); -- 2 слова
  select * from str_words_count('1,   2      ;3', ' ' ); -- 3 слова
*/

 Res = 0;

  -- PERFORM debuglog_clear();

  --PERFORM debuglog_add( 'Str', 'str_words_count', 'Sx', Sx ,'Старт');
  --PERFORM debuglog_add( 'Str', 'str_words_count', 'sUnChar ', sUnChar,'Старт');
  --PERFORM debuglog_add( 'Str', 'str_words_count', 'sXrenovina ', sXrenovina ,'Старт');


 IF (YesTrimBefore) THEN
    Sx = btrim(Sx,sUnChar);
    --PERFORM debuglog_add( 'Str', 'str_words_count', 'Sx ', Sx ,'Sx = btrim(Sx,sUnChar);');
 END IF;
 m = char_length(Sx);
 IF (m>0) THEN
   IF (char_length(sUnChar)>0) THEN
        -- ************************************
        -- Это - на предмет НеЧеткости описАния и работы какой-то из функций: substring, position
        IF char_length(sXrenovina)>0 THEN
            k = position(' ' in sUnChar);
            IF k<=0 THEN
                Sx = replace(Sx,' ',sXrenovina);
            END IF;
        END IF;
        -- ************************************

        -- PERFORM debuglog_add( 'Str', 'str_words_count', 'Sx ', Sx ,'После: sXrenovina');

        Res = 0;

        inword = false;
        i = 0;
        WHILE i<m LOOP
          i = i + 1;
          Ch = substring(Sx,i,1);
```

```
            k = position(Ch in sUnChar);
            IF k<=0 THEN
                IF (NOT inword) THEN Res = Res+1; END IF;
                inword = true;
            ELSE
                inword = false;
            END IF;

          END LOOP;

      ELSE
          Res = 1;
      END IF;
  END IF;

  -- PERFORM    debuglog_add( 'Str', 'str_words_count', 'Res', '' || Res ,'RETURN Res;');

  RETURN Res;
END;
$$ LANGUAGE plpgsql;
```

str_word_by_num

```
-- Получить слово о номеру из строки Sx

-- select * from debuglog_view('Str','str_word_by_num');
-- PERFORM debuglog_clear();

CREATE OR REPLACE FUNCTION str_word_by_num(Sx text, Num integer, sUnChar text DEFAULT ' ', YesTrimBefore boolean DEFAULT true, sXrenovina text DEFAULT '|') RETURNS text AS $$
DECLARE
      i integer;
      k integer;
      m integer;
      nWord integer;
      inword boolean;
      YesExit boolean;
      ch char(1);
      Res text;
BEGIN
/*
2016.12.18 Получить слово о номеру из строки Sx.
Разделителями слов могут быть любые символы из sUnChar
Параметры:
   Sx - входная строка;
   Num - номер слова в строке;
   sUnChar - НеСимволы. Строка (типа - множество НеСимволов);
   YesTrimBefore - если TRUE, то перед обработкой удаляются НеСимволы   слева и справа в Sx
```

sXrenovina - символ, на который заменяются пробелы (следствие НеЯсности для меня работы функций substring, position). Такого символа в принципе не должно быть в строке

Возвращаемое значение: слово по его номеру

Примеры:
```
  select str_word_by_num('   ,,,1, ;;; 2, 3', ' ,');   -- 4 слова
  select * from str_word_by_num('1,2 ;3', ';, ');   -- 3 слова
  select * from str_word_by_num('1,2 ;3', ','); -- 2 слова
  select * from str_word_by_num('1,2 ;3', ';3' ); -- 1 слово
  select * from str_word_by_num('1,2      ;3', ' ' ); -- 2 слова
  select * from str_word_by_num('1,   2      ;3', ' ' ); -- 3 слова
*/


 Res = '';

  -- PERFORM debuglog_clear();

  --PERFORM debuglog_add( 'Str', 'str_word_by_num', 'Sx', Sx ,'Старт');
  --PERFORM debuglog_add( 'Str', 'str_word_by_num', 'sUnChar ', sUnChar,'Старт');
  --PERFORM debuglog_add( 'Str', 'str_word_by_num', 'sXrenovina ', sXrenovina ,'Старт');
  --PERFORM debuglog_add( 'Str', 'str_word_by_num', 'Num ', '' || Num ,'Старт');


 IF (YesTrimBefore) THEN
     Sx = btrim(Sx,sUnChar);
     --PERFORM debuglog_add( 'Str', 'str_word_by_num', 'Sx ', Sx ,'Sx = btrim(Sx,sUnChar);');
 END IF;

 m = char_length(Sx);

 IF (m>0) THEN
    IF (char_length(sUnChar)>0) THEN
         -- ************************************
         -- Это - на предмет НеЧеткости описАния и работы какой-то из функций: substring, position
         IF char_length(sXrenovina)>0 THEN
             k = position(' ' in sUnChar);
             IF k<=0 THEN
                 Sx = replace(Sx,' ',sXrenovina);
             END IF;
         END IF;
         -- ************************************

         --PERFORM debuglog_add( 'Str', 'str_word_by_num', 'Sx ', Sx ,'После: sXrenovina');

         YesExit = false;
         nWord = 0;
         inword = false;
         i = 0;
         WHILE i<m LOOP
           i = i + 1;
           IF (not YesExit) THEN
```

```
                Ch = substring(Sx,i,1);
                k = position(Ch in sUnChar);
                IF k<=0 THEN
                    IF (NOT inword) THEN nWord = nWord+1; END IF;
                    inword = true;
                ELSE
                    inword = false;
                    IF (nWord>=Num) THEN
                        YesExit = true; -- Выход из цикла (надо почитать про exit;)
                    END IF;
                END IF;
                IF (inword) THEN
                    IF (nWord=Num) THEN
                        Res = Res || Ch;
                    END IF;
                END IF;
            END IF;
        END LOOP;
        Res = replace(Res, sXrenovina, ' ');
    ELSE
        Res = Sx;
    END IF;
 END IF;

  --PERFORM   debuglog_add( 'Str', 'str_word_by_num', 'Res', Res ,'RETURN Res;');

 RETURN Res;
END;
$$ LANGUAGE plpgsql;
```

param_val_from_str

```
-- Выделить значения параметра по его имени

CREATE OR REPLACE FUNCTION param_val_from_str(sParams text, sVar text, sVarDef text default
'', YesTrim boolean DEFAULT true) RETURNS text AS $$
BEGIN
/*
2016.12.16     Выделить значения параметра по его имени
  sParams - строка вида: 'x=1.25555;y=2;z=kkk;d=555;Sx="Привет, как дела?";S=Привет, как дела?'
  sVar - имя параметра, значение которого нужно получить
  sVarDef - значение по умолчанию (не реализовано в данной версии)
  YesTrim - если TRUE, то в возвращаемом значении удаляются пробелы слева и справа
Возвращаемое функцией значение: значение параметра (text)
Прим:
  1. Разделитель параметров - символ ";"
  2. Разделитель имени параметра и его значения - символ "="
  3. Имя переменной может быть в любом регистре
```

4. Если в значении используется символ "одинарные кавычки" (#39), то его надо дублировать (как это принято в SQL)
5. Запрещено:
5.1 использовать символы ";" и "=" внутри имен и значений параметров
5.2 между именем параметра и символом "=" использовать любые другие символы (т.е., вот это 'a =1.244;' - плохо (надо 'a=1.244;')
5.3 между именем параметра и символом ";" использовать любые другие символы (т.е., вот это 'a=1.244; b=5' - плохо (надо 'a=1.244;b=5')

Примеры:
select * from param_val_from_str('x=1.25555;y=2;z=kkk;d=555', 'z');
select * from param_val_from_str('x=1.25555;y=2;z=kkk;d=555;Sx="Привет, как дела ?"','Sx');
select * from param_val_from_str('x=1.25555;y=2;z=kkk;d=555;Sx= Привет, как дела ?','Sx');
select * from param_val_from_str('x=1.25555;y=2;z=kkk;d=555;Sx= Привет, как дела ?','Sx',",false);
select * from param_val_from_str('Коэфф-1=1.25555;y=2;z=kkk;Коэфф-2=2.2;Sx= Привет, как дела ?','Коэфф-2');
select * from param_val_from_str('Коэфф-1=1.25555;y=2;z=kkk;Коэфф-2=2.2;Sx= Привет, "Друг", как дела ?','Sx'); -- одинарные кавычки сдвоены (перед и после слова Друг)
*/

```
 IF (YesTrim) THEN
    RETURN   btrim((select Res[2] from string_to_array((select * from unnest( string_to_array(sParams,';') ) as r where upper(r) like upper(btrim(sVar)) || '=%'),'=') as Res));
 ELSE
    RETURN (select Res[2] from string_to_array((select * from unnest( string_to_array(sParams,';') ) as r where upper(r) like upper(btrim(sVar)) || '=%'),'=') as Res);
 END IF;
END;
$$ LANGUAGE plpgsql;
```

param_val_replace_

Заменить значение параметра по его имени в строке параметров

CREATE OR REPLACE FUNCTION **param_val_replace_**(sParams text, sVar text, sValNew text)
RETURNS text AS $$
BEGIN
/*
2016.12.16 Предназначена для вызова из param_val_replace()

Заменить значение параметра по его имени
sParams - строка вида: 'x=1.25555;y=2;z=kkk;d=555;Sx=Привет, как дела ?'
sVar - имя параметра
sValNew - новое значение

Возвращаемое значение: sParams с замененным значением для sVar
Прим.
Возвращаемое значение (sParams): слева и справа - символы {}.
Чтобы это исключить - следует вызывать функцию param_val_replace()

См., также, описание функции: ParamValFromStr

Примеры:
  select * from param_val_replace_('x=1.25555;y=2;z=kkk;d=bbb;Sx=Привет, как дела ?', 'Sx', 'Отлично!');
  select * from param_val_replace_('x=1.25555;y=2;z=kkk;d=bbb;Sx= Привет, как дела ?', 'Sx', 'Отлично!');
  select * from param_val_replace_('x=1.25555;y=2;z=kkk;d=bbb;Sx= Привет, как дела ?', 'Sx', '');
  select * from param_val_replace_('x=1.25555;y=2;z=kkk;d=bbb;Sx= Привет, как дела ?', 'd', '16.12.2016');
  select * from param_val_replace_('КоэффКакойто=1.25555;y=2;z=kkk;d=bbb;Sx= Привет, как дела ?', 'КоэффКакойто', '2.1');
  select * from param_val_replace_(ParamValReplace_('x=1.25555;y=2;z=kkk;d=bbb;Sx= Привет, как дела ?', 'Sx', '111'));
*/

  RETURN (select * from array_replace(string_to_array(sParams,';'),sVar || '=' || param_val_from_str(sParams, sVar, '', false),sVar || '=' || sValNew));

END;
$$ LANGUAGE plpgsql;

param_val_replace

Заменить значение параметра по его имени в строке параметров

CREATE OR REPLACE FUNCTION **param_val_replace**(sParams text, sVar text, sValNew text)
RETURNS text AS $$
BEGIN
/*
2016.12.16 Заменить значение параметра по его имени
  sParams - строка вида: 'x=1.25555;y=2;z=kkk;d=555;Sx=Привет, как дела ?'
  sVar - имя параметра
  sValNew - новое значение

См., также, описание функции: param_val_from_str

Примеры:
  select * from param_val_replace('x=1.25555;y=2;z=kkk;d=bbb;Sx=Привет, как дела ?', 'Sx', 'Отлично!');
  select * from param_val_replace('x=1.25555;y=2;z=kkk;d=bbb;Sx= Привет, как дела ?', 'Sx', 'Отлично!');
  select * from param_val_replace('x=1.25555;y=2;z=kkk;d=bbb;Sx= Привет, как дела ?', 'Sx', '');
  select * from param_val_replace('x=1.25555;y=2;z=kkk;d=bbb;Sx= Привет, как дела ?', 'd', '16.12.2016');
  select * from param_val_replace('КоэффКакойто=1.25555;y=2;z=kkk;d=bbb;Sx= Привет, как дела ?', 'КоэффКакойто', '2.1');
  select * from char_length(param_val_replace('x=1.25555;y=2;z=kkk;d=bbb;Sx= Привет, как дела ?', 'Sx', '111'));
*/

```
    RETURN (select * from btrim(param_val_replace_(sParams, sVar, sValNew),'{}'));

END;
$$ LANGUAGE plpgsql;
```

uuid_to_str

-- Конвертировать uuid в строку (text).
См. **uuid_to_str.**

Числа

Функции:
- int_is_null
- bigint_is_null

int_is_null

-- Проверка integer на null. Если null, то значение меняется на sdef

```
CREATE OR REPLACE FUNCTION int_is_null(
      sx integer,
      sdef integer DEFAULT 0)
  RETURNS integer AS
$$
BEGIN
  IF (sx is NULL) THEN
        IF (sdef is NULL) THEN sdef=0; END IF;
        RETURN sdef;
  ELSE
     RETURN sx;
  END IF;
END;
$$
  LANGUAGE plpgsql;
```

bigint_is_null

-- Проверка bigint на null. Если null, то значение меняется на sdef

```
CREATE OR REPLACE FUNCTION bigint_is_null(
      sx bigint,
      sdef bigint DEFAULT 0)
  RETURNS bigint AS
$$
BEGIN
  IF (sx is NULL) THEN
        IF (sdef is NULL) THEN sdef=0; END IF;
        RETURN sdef;
  ELSE
     RETURN sx;
  END IF;
END;
$$
  LANGUAGE plpgsql
```

UUID

Чтобы функции работы с uuid были доступны, следует выполнить команду:
create extension "uuid-ossp";

Функции:
- uuid_is_null
- uuid_to_str

uuid_is_null

Проверка uuid на null. Если null, то значение меняется на sdef

```
-- create extension "uuid-ossp";

CREATE OR REPLACE FUNCTION uuid_is_null(
      sx uuid,
      sdef uuid DEFAULT uuid_nil())
  RETURNS uuid AS
$$
BEGIN
  -- Прежде, чем использовать uuid, нужно выполнить: create extension "uuid-ossp";
  -- select uuid_is_null(null)
  -- select uuid_is_null(uuid_generate_v4())

  IF (sx is NULL) THEN
       IF (sdef is NULL) THEN sdef=uuid_nil(); END IF;
       RETURN sdef;
  ELSE
     RETURN sx;
  END IF;
END;
$$
  LANGUAGE plpgsql
```

uuid_to_str

```
-- Конвертировать uuid в строку (text)

CREATE OR REPLACE FUNCTION uuid_to_str(
      uuid_ uuid)
  RETURNS text AS
$$
BEGIN
```

```
  IF (uuid_ is NULL) THEN
       RETURN 'null';
  ELSE
    RETURN '' || uuid_;
  END IF;
END;
$$
  LANGUAGE plpgsql;
```

Логические

bool_is_null

-- Проверка boolean на null. Если null, то значение меняется на sdef

```
CREATE OR REPLACE FUNCTION bool_is_null(
      sx boolean,
      sdef boolean DEFAULT false)
  RETURNS boolean AS
$$
BEGIN
  IF (sx is NULL) THEN
        IF (sdef is NULL) THEN sdef=false; END IF;
        RETURN sdef;
  ELSE
     RETURN sx;
  END IF;
END;
$$
  LANGUAGE plpgsql
```

Отладка (debug)

Функции:
- debuglog_add/6
- debuglog_add/5
- debuglog_del/3
- debuglog_del/2
- debuglog_del/1
- debuglog_clear
- debuglog_view/3
- debuglog_view/2
- debuglog_view/1
- debuglog_view/0

debuglog_add/6

-- Добавить строку в таблицу debuglog

```
CREATE OR REPLACE FUNCTION debuglog_add
( thread_name_ character varying(100),
  func_name_ character varying(100),
  var_name_ character varying(100),
  icycle_ integer,
  var_val_ text,
  note_ text
)
RETURNS text AS
$$
DECLARE id_ uuid;

BEGIN
  /*   select * from debuglog_add( 'Поток-1', 'ФФФ-1', 'var-1', 5, '1.255', 'Прим-1');   */
  id_ = uuid_generate_v4();
  INSERT INTO debuglog
  (
    id,
    thread_name,
    func_name,
    var_name,
    icycle,
    var_val,
    note
  )
  VALUES
  (
    id_,
    thread_name_,
    func_name_,
```

```
    var_name_,
    icycle_,
    var_val_,
    note_
  );
  RETURN uuid_to_str(id_);
END;
$$ LANGUAGE plpgsql;
```

debuglog_add/5

-- Добавить строку в таблицу debuglog

```
CREATE OR REPLACE FUNCTION debuglog_add
( thread_name_ character varying(100),
  func_name_ character varying(100),
  var_name_ character varying(100),
  var_val_ text,
  note_ text
)
RETURNS text AS
$$
DECLARE id_ uuid;

BEGIN
  /*   select * from debuglog_add( 'Поток-1', 'ФФФ-1', 'var-1','1.255','Прим-1');   */
  /* PERFORM debuglog_add( 'Str', 'Str_WordsCount', 'Sx ', Sx ,'Старт');   */
  id_ = uuid_generate_v4();
  INSERT INTO debuglog
  (
    id,
    thread_name,
    func_name,
    var_name,
    var_val,
    note
  )
  VALUES
  (
    id_,
    thread_name_,
    func_name_,
    var_name_,
    var_val_,
    note_
  );
  RETURN uuid_to_str(id_);
END;
$$ LANGUAGE plpgsql;
```

debuglog_del/3

-- Удалить строки из таблицы debuglog

```
CREATE OR REPLACE FUNCTION debuglog_del
( thread_name_ character varying(100),
  func_name_ character varying(100),
  var_name_ character varying(100)
)
RETURNS boolean AS
$$
BEGIN
  /*   select * from debuglog_del( 'aaa', 'f1', 'v1');   */

  thread_name_ = btrim(str_is_null(thread_name_));
  func_name_ = btrim(str_is_null(func_name_));
  var_name_ = btrim(str_is_null(var_name_));

  DELETE FROM debuglog
  WHERE
    upper(thread_name) = upper(thread_name_)
    and
    upper(func_name) = upper(func_name_)
    and
    upper(var_name) = upper(var_name_);
  RETURN true;

END;
$$ LANGUAGE plpgsql;
```

debuglog_del/2

-- Удалить строки из таблицы debuglog

```
CREATE OR REPLACE FUNCTION debuglog_del(
    thread_name_ character varying,
    func_name_ character varying)
  RETURNS boolean AS
$$
BEGIN
  /*   select * from debuglog_del( 'aaa', 'f1');   */

  thread_name_ = btrim(str_is_null(thread_name_));
  func_name_ = btrim(str_is_null(func_name_));

  DELETE FROM debuglog
  WHERE
```

```
    upper(thread_name) = upper(thread_name_)
    and
    upper(func_name) = upper(func_name_);
  RETURN true;
END;
$$
  LANGUAGE plpgsql;
```

debuglog_del/1

-- Удалить строки из таблицы debuglog

```
CREATE OR REPLACE FUNCTION debuglog_del
( thread_name_ character varying(100)
)
RETURNS boolean AS
$$
BEGIN
  /*   select * from debuglog_del( 'Поток-1');   */

  thread_name_ = btrim(str_is_null(thread_name_));

  DELETE FROM debuglog
  WHERE
    upper(thread_name) = upper(thread_name_);
  RETURN true;
END;
$$ LANGUAGE plpgsql;
```

debuglog_clear

-- Полностью очистить таблицу debuglog

```
CREATE OR REPLACE FUNCTION debuglog_clear
()
RETURNS boolean AS
$$
BEGIN
  /*   select * from debuglog_clear();   */
  /* PERFORM debuglog_clear(); */

  DELETE FROM debuglog;

  RETURN true;
END;
$$ LANGUAGE plpgsql;
```

debuglog_view/3

-- Открыть (select) таблицу debuglog

```
CREATE OR REPLACE FUNCTION debuglog_view
(
  thread_name_ character varying(100),
  func_name_ character varying(100),
  var_name_ character varying(100)
)
RETURNS SETOF debuglog AS
$$
  /* SELECT * FROM   debuglog_view('Поток-1', 'ФФФ-1', 'var-2'); */
  SELECT
    *
  FROM
    debuglog
  WHERE
    upper(thread_name) = upper(thread_name_)
    and
    upper(func_name) = upper(func_name_)
    and
    upper(var_name) = upper(var_name_)
    order by npp;
$$ LANGUAGE sql;
```

debuglog_view/2

-- Открыть (select) таблицу debuglog

```
CREATE OR REPLACE FUNCTION debuglog_view
(
  thread_name_ character varying(100),
  func_name_ character varying(100)
)
RETURNS SETOF debuglog AS
$$
  /* SELECT * FROM   debuglog_view('Поток-1', 'ФФФ-1'); */
  SELECT
    *
  FROM
    debuglog
  WHERE
    upper(thread_name) = upper(thread_name_)
    and
    upper(func_name) = upper(func_name_)
```

```
    order by npp;
$$ LANGUAGE sql;
```

debuglog_view/1

```
-- Открыть (select) таблицу debuglog

CREATE OR REPLACE FUNCTION debuglog_view
(
  thread_name_ character varying(100)
)
RETURNS SETOF debuglog AS
$$
  /* SELECT * FROM   debuglog_view('Поток-1'); */
  SELECT
    *
  FROM
    debuglog
  WHERE
    upper(thread_name) = upper(thread_name_)
    order by npp;
$$ LANGUAGE sql;
```

debuglog_view/0

```
-- Открыть (select) таблицу debuglog

CREATE OR REPLACE FUNCTION debuglog_view
(
)
RETURNS SETOF debuglog AS
$$
  /* SELECT * FROM   debuglog_view(); */
  /* SELECT * FROM   debuglog_view() order by dt; */
  SELECT
    *
  FROM
    debuglog
    order by thread_name, npp;
$$ LANGUAGE sql;
```

Триггерные функции

- Таблица debuglog
- Таблицы tree2 и tree2p
- Таблица tree1

Таблица debuglog

Таблица debuglog

Триггерные функции:
- debuglog_func_trig_bi

debuglog_func_trig_bi

```
-- Триггерная функция для   debuglog

CREATE OR REPLACE FUNCTION debuglog_func_trig_bi()
  RETURNS trigger AS
$$
DECLARE n integer;
BEGIN
  NEW.thread_name = btrim(str_is_null(NEW.thread_name));
  NEW.func_name = btrim(str_is_null(NEW.func_name));
  NEW.var_name = btrim(str_is_null(NEW.var_name));
  IF (NEW.note IS NOT NULL) THEN NEW.note = btrim(NEW.note); END IF;
  NEW.icycle = int_is_null(NEW.icycle);

  -- NEW.dt = now();
  --NEW.dt = date_trunc('sec',now());
  NEW.dt = to_char(NOW(),'YYYY.MM.DD HH24:MI:SS'); -- http://programming-lang.com/html/sql/glava%205/index14.htm


  IF (NEW.npp IS NULL) THEN
     SELECT MAX(npp) from debuglog where thread_name = NEW.thread_name into n;
     n = int_is_null(n);
     NEW.npp = n+1;
  END IF;

  -- Прежде, чем использовать uuid, нужно выполнить: create extension "uuid-ossp";
  IF (NEW.id IS NULL) THEN
     NEW.id = uuid_generate_v4();
  END IF;

  return NEW;
```

```
END;
$$
  LANGUAGE plpgsql
```

Таблицы tree2 и tree2p

Таблицы tree2 и tree2p

Триггерные функции:
- tree2_func1_trig_bi
- tree2_func1_trig_ai
- tree2_func1_trig_bu
- tree2_func1_trig_au
- tree2_func1_trig_bd
- tree2p_func1_trig_bi
- tree2p_func1_trig_bu

tree2_func1_trig_bi

```
-- Триггерная функция для   tree2 (BI)

CREATE OR REPLACE FUNCTION tree2_func1_trig_bi()
  RETURNS trigger AS
$$
DECLARE ilev integer;
BEGIN
  NEW.yesmovep = false;
  NEW.yesmovec = false;
  NEW.id_parent = bigint_is_null(NEW.id_parent);
  NEW.level_hierarchy = 0;
  IF (NEW.id_parent>0) THEN
      SELECT level_hierarchy from tree2 where id=NEW.id_parent into ilev;
      ilev = int_is_null(ilev);
      NEW.level_hierarchy = ilev + 1;
  END IF;

  NEW.id = nextval('seq_tree2');

  return NEW;
END;
$$
  LANGUAGE plpgsql;
```

tree2_func1_trig_ai

```
-- Триггерная функция для   tree2 (AI)

CREATE OR REPLACE FUNCTION tree2_func1_trig_ai()
  RETURNS trigger AS
$$
BEGIN

  IF (NEW.id_parent>0) THEN

     PERFORM tree2p_copy_from_1_to_2(NEW.id_parent, NEW.id, false);

     INSERT INTO tree2p
     (
       id_owner,
       id_parent,
       level_hierarchy
     )
     VALUES
     (
       NEW.id,
       NEW.id_parent,
       NEW.level_hierarchy
     );
  END IF;

  return NEW;
END;
$$
  LANGUAGE plpgsql;
```

tree2_func1_trig_bu

```
-- Триггерная функция для   tree2 (BU)

CREATE OR REPLACE FUNCTION tree2_func1_trig_bu()
  RETURNS trigger AS
$$
DECLARE ilev integer;
DECLARE yesm boolean;
BEGIN
  NEW.id = OLD.id;
  NEW.yesmovec = false;
  yesm = false;
  IF NEW.yesmovep THEN yesm = TRUE; END IF;
  NEW.id_parent = bigint_is_null(NEW.id_parent);
  IF (NEW.id_parent<>OLD.id_parent) THEN yesm = TRUE; END IF;
```

```
  IF (yesm) THEN
     NEW.yesmovec = true;
     NEW.level_hierarchy = 0;
     IF (NEW.id_parent>0) THEN
        SELECT level_hierarchy from tree2 where id=NEW.id_parent into ilev;
        ilev = int_is_null(ilev);
        NEW.level_hierarchy = ilev + 1;
     END IF;
  END IF;

  NEW.yesmovep = false;
  return NEW;
END;
$$
  LANGUAGE plpgsql;
```

tree2_func1_trig_au

```
-- Триггерная функция для   tree2 (AU)

CREATE OR REPLACE FUNCTION tree2_func1_trig_au()
  RETURNS trigger AS
$$
BEGIN
  IF NEW.yesmovec THEN
     -- **********************************
     -- Предварительно все удаляем (мож и лишнее)...
     DELETE
     FROM
       tree2p
     WHERE
       tree2p.id_owner = NEW.id;
     -- **********************************

     IF (NEW.id_parent>0) THEN

        PERFORM tree2p_copy_from_1_to_2(NEW.id_parent, NEW.id, false);

        INSERT INTO tree2p
        (
          id_owner,
          id_parent,
          level_hierarchy
        )
        VALUES
        (
          NEW.id,
          NEW.id_parent,
          NEW.level_hierarchy
```

```
        );

        UPDATE
          tree2
        SET
          yesmovep = true     /* Принудительно обновляем ближайшие дочерние */
        WHERE
          tree2.id_parent=NEW.id;

    END IF;

  END IF;

  return NEW;
END;
$$
  LANGUAGE plpgsql;
```

tree2_func1_trig_bd

```
-- Триггерная функция для   tree2 (BD)

CREATE OR REPLACE FUNCTION tree2_func1_trig_bd()
  RETURNS trigger AS
$$
BEGIN

    DELETE
    FROM
      tree2
    WHERE
      tree2.id_parent = OLD.id;

  return OLD;
END;
$$
  LANGUAGE plpgsql;
```

tree2p_func1_trig_bi

```
-- Триггерная функция для   tree2p (BI)

CREATE OR REPLACE FUNCTION tree2p_func1_trig_bi()
  RETURNS trigger AS
$$
BEGIN
```

```
  NEW.id = nextval('seq_tree2p');

  return NEW;
END;
$$
  LANGUAGE plpgsql;
```

tree2p_func1_trig_bu

```
-- Триггерная функция для  tree2p (BU)

CREATE OR REPLACE FUNCTION tree2p_func1_trig_bu()
  RETURNS trigger AS
$$
BEGIN

  NEW.id = OLD.id;

  return NEW;
END;
$$
  LANGUAGE plpgsql;
```

Таблица tree1

Таблица tree1

Триггерные функции:
- tree1_func1_trig_bi
- tree1_func1_trig_bu
- tree1_func1_trig_au
- tree1_func1_trig_bd

tree1_func1_trig_bi

```
-- Триггерная функция для  tree1 (BI)

CREATE OR REPLACE FUNCTION tree1_func1_trig_bi()
  RETURNS trigger AS
$$
BEGIN
```

```
  NEW.yesmovep = false;
  NEW.id_parent = bigint_is_null(NEW.id_parent);
  NEW.level_hierarchy = 0;
  NEW.parents_id='';
  IF (NEW.id_parent>0) THEN
      SELECT level_hierarchy, parents_id from tree1 where id=NEW.id_parent into NEW.level_hierarchy,
NEW.parents_id;
      NEW.level_hierarchy = int_is_null(NEW.level_hierarchy);
      NEW.level_hierarchy = NEW.level_hierarchy + 1;
      NEW.parents_id = btrim(str_is_null(NEW.parents_id));
      NEW.parents_id = NEW.parents_id || '[' || NEW.id_parent || ']';
  END IF;

  NEW.id = nextval('seq_tree1');

  return NEW;
END;
$$
  LANGUAGE plpgsql;
```

tree1_func1_trig_bu

```
-- Триггерная функция для  tree1 (BU)

CREATE OR REPLACE FUNCTION tree1_func1_trig_bu()
  RETURNS trigger AS
$$
BEGIN
  NEW.id = OLD.id;

  NEW.id_parent = bigint_is_null(NEW.id_parent);
  NEW.level_hierarchy = 0;
  NEW.parents_id='';
  IF (NEW.id_parent>0) THEN
      SELECT level_hierarchy, parents_id from tree1 where id=NEW.id_parent into NEW.level_hierarchy,
NEW.parents_id;
      NEW.level_hierarchy = int_is_null(NEW.level_hierarchy);
      NEW.level_hierarchy = NEW.level_hierarchy + 1;
      NEW.parents_id = btrim(str_is_null(NEW.parents_id));
      NEW.parents_id = NEW.parents_id || '[' || NEW.id_parent || ']';
  END IF;

  NEW.yesmovep = false;

  return NEW;
END;
$$
  LANGUAGE plpgsql;
```

tree1_func1_trig_au

-- Триггерная функция для tree1 (AU)

```
CREATE OR REPLACE FUNCTION tree1_func1_trig_au()
  RETURNS trigger AS
$$
BEGIN
  IF (NEW.parents_id <> OLS.parents_id) THEN
     UPDATE
       tree1
     SET
       yesmovep = true     /* Принудительно обновляем ближайшие дочерние */
     WHERE
       tree1.id_parent=NEW.id;
  END IF;
  return NEW;
END;
$$
  LANGUAGE plpgsql;
```

tree1_func1_trig_bd

-- Триггерная функция для tree1 (BD)

```
CREATE OR REPLACE FUNCTION tree1_func1_trig_bd()
  RETURNS trigger AS
$$
BEGIN

    DELETE
    FROM
      tree1
    WHERE
      tree1.id_parent = OLD.id;

  return OLD;
END;
$$
  LANGUAGE plpgsql;
```

Специальные

"Деревья"

- Вариант-1: "Предки в этой же таблице в отдельном поле"
- Вариант-2. "Предки в отдельной таблице"

Вариант-1: "Предки в этой же таблице в отдельном поле"

tree1_parents_get

Получить всех предков для child_id, начиная с самого древнего по таблице tree1

```
CREATE OR REPLACE FUNCTION tree1_parents_get (child_id bigint, lev_min integer DEFAULT 0,
lev_max integer DEFAULT 999999)
RETURNS    SETOF bigint AS
$$
/*
Получить всех предков для child_id, начиная с самого древнего, уровень иерархии которых лежит в
заданном интервале level_hierarchy
Примеры:
      select * from tree1_parents_get(6);
      select
        tpg,
        (select name_this from tree1 where id=tpg) parent_name,
        (select level_hierarchy from tree1 where id=tpg) lev_name
      from
      tree1_parents_get(6) tpg;
*/
DECLARE
  c integer;
  i integer;
  S text;
  id_p bigint;
  lev integer;
  pars_id text;
BEGIN
  child_id = bigint_is_null(child_id);
  IF child_id>0 THEN
      lev_min = int_is_null(lev_min);
      IF lev_min<0 THEN lev_min=0; END IF;
      lev_max = int_is_null(lev_max);
      IF lev_max<lev_min THEN lev_max=lev_min; END IF;
      SELECT parents_id FROM tree1 WHERE id = child_id into pars_id;
```

```
      pars_id = str_is_null(pars_id);
      IF pars_id<>'' THEN
         c = str_words_count(pars_id,'[]');
         IF c>0 THEN
            --i = c; -- если нужна инверсия
            i=0;
            --WHILE i>0 LOOP -- если нужна инверсия
            WHILE i<c LOOP
              i = i + 1;
              S = str_word_by_num(pars_id, i, '[]');
              id_p = S;   -- нужен более безопасный путь преобразования
              IF id_p > 0 THEN
                 SELECT level_hierarchy FROM tree1 WHERE id = id_p   into lev;
                 lev = int_is_null(lev);
                 IF (lev>=lev_min) AND (lev<=lev_max) THEN
                    RETURN NEXT id_p;
                 END IF;
              END IF;
              --i = i - 1; -- если нужна инверсия
            END LOOP;
         END IF;
      END IF;
  END IF;
  RETURN;
END;
$$
LANGUAGE plpgsql
```

Вариант-2. "Предки в отдельной таблице"

- tree2p_copy_from_1_to_2
- tree2_childs_get
- tree2_parents_get

tree2p_copy_from_1_to_2

-- Копирует всех "предков" в таблице tree2p от idowner_from к idowner_to.
-- Используется в триггерных функциях таблицы tree2

```
CREATE OR REPLACE FUNCTION tree2p_copy_from_1_to_2(idowner_from bigint, idowner_to bigint,
yesclearbefore boolean DEFAULT false)
  RETURNS integer AS
$$
DECLARE
  res integer;
```

```
  id_p bigint;
  id_ bigint;
  lev_h integer;
  ccc CURSOR FOR select id_parent, level_hierarchy from tree2p where id_owner = idowner_from order
by level_hierarchy;

BEGIN
    /*
     Копирует всех "родителей" в таблице tree2p от idowner_from к idowner_to.
     Вызывается из:   tree2_func1_trig_ai(); tree2_func1_trig_au().
    */
    res = 0;
    yesclearbefore = bool_is_null(yesclearbefore);
    idowner_from = bigint_is_null(idowner_from);
    IF (idowner_from>0) THEN
        idowner_to = bigint_is_null(idowner_to);
        IF (idowner_to>0) THEN
            IF (idowner_from<>idowner_to) THEN
                SELECT id FROM tree2 WHERE id=idowner_to INTO id_;
                id_ = bigint_is_null(id_);
                IF (id_ > 0) THEN

                    IF yesclearbefore THEN
                        DELETE
                        FROM
                          tree2p
                        WHERE
                         (id_owner=idowner_to);
                    END IF;

                    OPEN ccc;
                    LOOP
                      FETCH ccc INTO id_p, lev_h;

                      IF NOT FOUND THEN EXIT;END IF;

                      id_p = bigint_is_null(id_p);
                      lev_h = int_is_null(lev_h);

                      res = res + 1; -- Выяснить, как оценить кол-во записей, реально обработанных
UPDATE, INSERT, DELETE

                      SELECT id_parent FROM tree2p WHERE (id_owner=idowner_to) and (id_parent=id_p)
INTO id_;
                      id_ = bigint_is_null(id_);
                      IF (id_ > 0) THEN
                        UPDATE
                          tree2p
                        SET
                          level_hierarchy = lev_h
                        WHERE
                          (id_owner=idowner_to) and (id_parent=id_p);
                      ELSE
```

```
                    INSERT INTO tree2p
                    (
                        id_owner,
                        id_parent,
                        level_hierarchy
                    )
                    VALUES
                    (
                        idowner_to,
                        id_p,
                        lev_h
                    );

                END IF;

            END LOOP;
            CLOSE ccc;


        END IF;
    END IF;
  END IF;
END IF;

  RETURN res;
END;
$$
  LANGUAGE plpgsql;
```

tree2_childs_get

-- Отбор дочерних для предка ancestor_id, уровень иерархии которых находится в пределах lev_min ... lev_max по таблицам tree2 и tree2p

```
CREATE OR REPLACE FUNCTION tree2_childs_get (ancestor_id bigint, lev_min integer DEFAULT 1, lev_max integer DEFAULT 9999)
  --RETURNS TABLE (id_parent bigint, id bigint, level_hierarchy integer) AS
  RETURNS TABLE (id bigint) AS
$$
  -- Отбор дочерних для предка ancestor_id, уровень иерархии которых находится в пределах lev_min ... lev_max
  -- select   id, (select name_this from tree2 where id=tcg.id), (select level_hierarchy from tree2 where id=tcg.id) from tree2_childs_get(1) tcg
  -- select   id, (select name_this from tree2 where id=tcg.id), (select level_hierarchy from tree2 where id=tcg.id) from tree2_childs_get(3) tcg

  SELECT
    id
```

```
    --*
    --id_parent,
    --id,
    --level_hierarchy
  FROM
    tree2
  WHERE
    (
    level_hierarchy >= lev_min
    AND
    level_hierarchy <= lev_max
    )
    AND
    (
    id IN (SELECT id_owner FROM tree2p WHERE id_parent = ancestor_id)
    )
    ORDER BY
      id_parent, level_hierarchy;
$$
  LANGUAGE 'sql'
```

tree2_parents_get

Получить всех предков для child_id, начиная с самого древнего по таблицам tree2 и tree2p

```
CREATE OR REPLACE FUNCTION tree2_parents_get(child_id bigint)
  RETURNS SETOF bigint AS
$BODY$
DECLARE
  id_p bigint;
  ccc CURSOR FOR select id_parent from tree2p where id_owner = child_id order by level_hierarchy;
BEGIN
  /*
    Получить всех предков для child_id, начиная с самого древнего
    select * from tree2_parents_get(6);
    select
      tpg,
      (select name_this from tree2 where id=tpg) parent_name,
      (select level_hierarchy from tree2 where id=tpg) lev_name
    from
    tree2_parents_get(6) tpg;
  */

  child_id = bigint_is_null(child_id);
  IF child_id>0 THEN

    OPEN ccc;
    LOOP
```

```
        FETCH ccc INTO id_p;

        IF NOT FOUND THEN EXIT;END IF;

        RETURN NEXT id_p;

      END LOOP;
      CLOSE ccc;

  END IF;
  RETURN; -- Необязательный
END
$BODY$
  LANGUAGE plpgsql
```

Таблицы

- Таблицы для отладки
- "Деревья"

Таблицы для отладки

- debuglog

debuglog

```
-- Используется для ведения протокола при отладке хранимок...

CREATE TABLE debuglog
(
  id uuid NOT NULL,
  thread_name character varying(100),   -- имя потока
  npp integer,                                    -- порядковый номер в потоке
  func_name character varying(100),       -- имя функции
  var_name character varying(100),         -- имя переменной
  icycle integer,                                 -- номер цикла
  var_val text,                                    -- номер значения
  note text,                                       -- примечание
  dt timestamp without time zone,
  CONSTRAINT pk_debuglog PRIMARY KEY (id)
)
WITH (
```

```
  OIDS=FALSE
);
ALTER TABLE debuglog
  OWNER TO postgres;

-- Index: debuglog_dt
-- DROP INDEX debuglog_dt;
CREATE INDEX debuglog_dt  ON debuglog  USING btree  (dt);

-- Index: debuglog_funcname
-- DROP INDEX debuglog_funcname;
CREATE INDEX debuglog_funcname  ON debuglog  USING btree  (func_name COLLATE
pg_catalog."default");

-- Index: debuglog_npp
-- DROP INDEX debuglog_npp;
CREATE INDEX debuglog_npp  ON debuglog  USING btree  (npp);

-- Index: debuglog_threadname
-- DROP INDEX debuglog_threadname;
CREATE INDEX debuglog_threadname  ON debuglog  USING btree  (thread_name COLLATE
pg_catalog."default");

-- Index: debuglog_varname
-- DROP INDEX debuglog_varname;
CREATE INDEX debuglog_varname  ON debuglog  USING btree  (var_name COLLATE
pg_catalog."default");

-- Index: debuglog_icycle
-- DROP INDEX debuglog_icycle;
CREATE INDEX debuglog_icycle
  ON debuglog
  USING btree
  (icycle);
```

Триггеры:

```
CREATE TRIGGER debuglog_bi
  BEFORE INSERT
  ON debuglog
  FOR EACH ROW
  EXECUTE PROCEDURE debuglog_func_trig_bi();
```

"Деревья"

Таблицы:

Вариант-1: "Предки в этой же таблице в отдельном поле"
- tree1

Вариант-2: "Предки в отдельной таблице"
- tree2
- tree2p

Вариант-3: "Предки в detail-таблице в отдельном поле"

Вариант-1: "Предки в этой же таблице в отдельном поле"

Позже...

tree1

Пример реализации дерева (вариант-1, предки в этой же таблице в отдельном поле)

```
CREATE SEQUENCE seq_tree1;

CREATE TABLE tree1
(
  id bigint NOT NULL,
  id_parent bigint NOT NULL,
  level_hierarchy integer NOT NULL,
  yesmovep boolean,
  parents_id text,
  name_this character varying(50),
  CONSTRAINT tree1_pkey PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
```

```
);
ALTER TABLE tree1
  OWNER TO postgres;

CREATE INDEX tree1_idparent
  ON tree1
  USING btree
  (id_parent);

CREATE INDEX tree1_levhier
  ON tree1
  USING btree
  (level_hierarchy);
```

Триггеры:

```
CREATE TRIGGER tree1_bi
 BEFORE INSERT
 ON tree1
 FOR EACH ROW
 EXECUTE PROCEDURE tree1_func1_trig_bi();

CREATE TRIGGER tree1_bu
 BEFORE UPDATE
 ON tree1
 FOR EACH ROW
 EXECUTE PROCEDURE tree1_func1_trig_bu();

CREATE TRIGGER tree1_au
 AFTER UPDATE
 ON tree1
 FOR EACH ROW
 EXECUTE PROCEDURE tree1_func1_trig_au();

CREATE TRIGGER tree1_bd
  BEFORE DELETE
  ON tree1
  FOR EACH ROW
  EXECUTE PROCEDURE tree1_func1_trig_bd();
```

Данные:

	id [PK] bigint	id_parent bigint	level_hierarchy integer	yesmovep boolean	parents_id text	name_this character varying(50)
1	1	0	0	FALSE	''	0a
2	2	0	0	FALSE	''	0b
3	3	1	1	FALSE	[1]	0a-1
4	4	3	2	FALSE	[1][3]	0a-1-1
5	5	3	2	FALSE	[1][3]	0a-1-2
6	6	4	3	FALSE	[1][3][4]	0a-1-1-1
*						

1;0;0;FALSE;"""";"0a"
2;0;0;FALSE;"""";"0b"
3;1;1;FALSE;"[1]";"0a-1"
4;3;2;FALSE;"[1][3]";"0a-1-1"
5;3;2;FALSE;"[1][3]";"0a-1-2"
6;4;3;FALSE;"[1][3][4]";"0a-1-1-1"

Вариант-2. "Предки в отдельной таблице"

Таблицы:
- tree2
- tree2p

Функции (специальные)

Триггерные функции

tree2

Пример реализации дерева (вариант-2, предки в отдельной таблице tree2p)

Основная таблица (дерево)

tree2 - "дерево"
tree2p - список всех "родителей"

CREATE SEQUENCE **seq_tree2**;

CREATE TABLE **tree2**
(
  id bigint NOT NULL,
  id_parent bigint NOT NULL,

```
  level_hierarchy integer NOT NULL,
  yesmovep boolean,
  yesmovec boolean,
  name_this character varying(50), -- для отладки и иллюстрации...
  CONSTRAINT tree2_pkey PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE tree2
  OWNER TO postgres;

CREATE INDEX tree2_idparent
  ON tree2
  USING btree
  (id_parent);

CREATE INDEX tree2_levhier
  ON tree2
  USING btree
  (level_hierarchy);
```

Триггеры:

```
CREATE TRIGGER tree2_bi
  BEFORE INSERT
  ON tree2
  FOR EACH ROW
  EXECUTE PROCEDURE tree2_func1_trig_bi();

CREATE TRIGGER tree2_ai
  AFTER INSERT
  ON tree2
  FOR EACH ROW
  EXECUTE PROCEDURE tree2_func1_trig_ai();

CREATE TRIGGER tree2_bd
  BEFORE DELETE
  ON tree2
  FOR EACH ROW
  EXECUTE PROCEDURE tree2_func1_trig_bd();

CREATE TRIGGER tree2_bu
  BEFORE UPDATE
  ON tree2
  FOR EACH ROW
  EXECUTE PROCEDURE tree2_func1_trig_bu();

CREATE TRIGGER tree2_au
  AFTER UPDATE
  ON tree2
  FOR EACH ROW
  EXECUTE PROCEDURE tree2_func1_trig_au();
```

Данные:

	id [PK] bigint	id_parent bigint	level_hierarchy integer	yesmovep boolean	yesmovec boolean	name_this character varying(50)
1	1	0	0	FALSE	FALSE	0a
2	2	0	0	FALSE	FALSE	0b
3	3	1	1	FALSE	TRUE	0a-1
4	4	3	2	FALSE	TRUE	0a-1-1
5	5	3	2	FALSE	TRUE	0a-1-2
6	6	4	3	FALSE	TRUE	0a-1-1-1
*						

1;0;0;FALSE;FALSE;"0a"
2;0;0;FALSE;FALSE;"0b"
3;1;1;FALSE;TRUE;"0a-1"
4;3;2;FALSE;TRUE;"0a-1-1"
5;3;2;FALSE;TRUE;"0a-1-2"
6;4;3;FALSE;TRUE;"0a-1-1-1"

tree2p

Пример реализации дерева (вариант-2)

Таблица "предков"

tree2 - "дерево"
tree2p - список всех "родителей"

CREATE SEQUENCE **seq_tree2p**;

CREATE TABLE **tree2p**
(
  id bigint NOT NULL,
  id_owner bigint NOT NULL,
  id_parent bigint NOT NULL,
  level_hierarchy integer NOT NULL,
  CONSTRAINT tree2p_pkey PRIMARY KEY (id),
  CONSTRAINT fk_tree2p_1 FOREIGN KEY (id_owner)
      REFERENCES tree2 (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE CASCADE
)
WITH (
  OIDS=FALSE
);
ALTER TABLE tree2p
  OWNER TO postgres;

CREATE INDEX tree2p_idowner

```
  ON tree2p
  USING btree
  (id_owner);

CREATE INDEX tree2p_idparent
  ON tree2p
  USING btree
  (id_parent);

CREATE INDEX tree2p_levhier
  ON tree2p
  USING btree
  (level_hierarchy);
```

Триггеры:

```
CREATE TRIGGER tree2p_bi
  BEFORE INSERT
  ON tree2p
  FOR EACH ROW
  EXECUTE PROCEDURE tree2p_func1_trig_bi();

CREATE TRIGGER tree2p_bu
  BEFORE UPDATE
  ON tree2p
  FOR EACH ROW
  EXECUTE PROCEDURE tree2p_func1_trig_bu();
```

Вариант-3: "Предки в detail-таблице в отдельном поле"

Позже...

Наследование деревьев

Отладка

Сценарий:

1. Внутри функции (по которой отладка) в самом ее начале выполняется:

```
PERFORM debuglog_clear();   -- очистить всю таблицу debuglog
```

2. Далее, в соответствующих точках выполняются соответствующие (которая подходят лучше) функции:

PERFORM debuglog_add(6 параметров) или PERFORM debuglog_add(5 параметров)

Например, так:

```
PERFORM debuglog_add( 'Str', 'str_words_count', 'Sx', Sx ,'Старт');
PERFORM debuglog_add( 'Str', 'str_words_count', 'sUnChar ', sUnChar,'Старт');
PERFORM debuglog_add( 'Str', 'str_words_count', 'sXrenovina ', sXrenovina ,'Старт');
PERFORM   debuglog_add( 'Str', 'str_words_count', 'Res', '' || Res ,'Перед RETURN Res;');
```

3. После выполнения функции (по которой отладка) вызывается соответствующая (которая подходит) функция:

```
select * from debuglog_view('Str','str_words_count');
```

Пример:

1. Функция, по которой отладка:

```
CREATE OR REPLACE FUNCTION str_words_count(Sx text, sUnChar text DEFAULT ' ',
YesTrimBefore boolean DEFAULT true, sXrenovina text DEFAULT '|') RETURNS integer AS $$
DECLARE
      i integer;
      k integer;
      m integer;
      inword boolean;
      ch char(1);
      Res integer;
BEGIN

 Res = 0;

   PERFORM debuglog_clear();

   PERFORM debuglog_add( 'Str', 'str_words_count', 'Sx', Sx ,'Старт');
   PERFORM debuglog_add( 'Str', 'str_words_count', 'sUnChar ', sUnChar,'Старт');
   PERFORM debuglog_add( 'Str', 'str_words_count', 'sXrenovina ', sXrenovina ,'Старт');

 IF (YesTrimBefore) THEN
     Sx = btrim(Sx,sUnChar);

     PERFORM debuglog_add( 'Str', 'str_words_count', 'Sx ', Sx ,'Sx = btrim(Sx,sUnChar);');

 END IF;
 m = char_length(Sx);
 IF (m>0) THEN
    IF (char_length(sUnChar)>0) THEN
          -- ************************************
          -- Это - на предмет НеЧеткости описАния и работы какой-то из функций: substring, position
          IF char_length(sXrenovina)>0 THEN
              k = position(' ' in sUnChar);
              IF k<=0 THEN
                  Sx = replace(Sx,' ',sXrenovina);
              END IF;
          END IF;
          -- ************************************

           PERFORM debuglog_add( 'Str', 'str_words_count', 'Sx ', Sx ,'После: sXrenovina');

          Res = 0;

          inword = false;
          i = 0;
          WHILE i<m LOOP
            i = i + 1;
            Ch = substring(Sx,i,1);
            k = position(Ch in sUnChar);
            IF k<=0 THEN
```

```
            IF (NOT inword) THEN Res = Res+1; END IF;
            inword = true;
        ELSE
            inword = false;
        END IF;

    END LOOP;

  ELSE
    Res = 1;
  END IF;
END IF;

 PERFORM  debuglog_add( 'Str', 'str_words_count', 'Res', '' || Res ,'Перед RETURN Res;');

 RETURN Res;
END;
$$ LANGUAGE plpgsql;
```

2. Выполняется отлаживаемая функция:

```
select str_words_count('   ,,,1, ;;; 2, 3', ' ,');   -- 4 слова
```

3. Просмотр Log`a:

```
select * from debuglog_view('Str','str_words_count');
```