----------------------------------------------------

--Вывести имя функции и типы принимаемых значений

----------------------------------------------------

create or replace procedure show\_functions() as

$$

declare

cur cursor

for select proname, proargtypes

from (

select proname, pronargs, prorettype, proargtypes

from pg\_proc

where pronargs > 0

) AS tmp;

row record;

begin

open cur;

loop

fetch cur into row;

exit when not found;

raise notice '{func\_name : %} {args : %}', row.proname, row.proargtypes;

end loop;

close cur;

end

$$ language plpgsql;

call show\_functions();

----------------------------------------------------

--Создать хранимую процедуру с выходным параметром, которая выводит

--список имен и параметров всех скалярных SQL функций пользователя

--(функции типа 'FN') в текущей базе данных. Имена функций без параметров

--не выводить. Имена и список параметров должны выводиться в одну строку.

--Выходной параметр возвращает количество найденных функций.

--Созданную хранимую процедуру протестировать.

----------------------------------------------------

CREATE PROCEDURE MyTables13

@cnt int OUTPUT

AS

BEGIN

SELECT \* FROM sys.all\_objects

where type = 'FN'

SELECT @cnt = COUNT(\*) FROM sys.all\_objects

where type = 'FN'

END

GO

DECLARE @cnt int

EXEC MyTables13 @cnt output

GO

PRINT 'Êîëè÷åñòâî ôóíêöèé: ' + CONVERT(VARCHAR, @cnt)

----------------------------------------------------

-- Создать хранимую процедуру с входным параметром – имя базы данных,

-- которая выводит имена ограничений CHECK и выражения SQL, которыми

-- определяются эти ограничения CHECK, в тексте которых на языке SQL

-- встречается предикат 'LIKE'. Созданную хранимую процедуру

-- протестировать.

----------------------------------------------------

create extension dblink;

create or replace procedure get\_like\_constraints(in data\_base\_name text)

language plpgsql

as

$$

declare

constraint\_rec record;

begin

for constraint\_rec in select \*

from dblink(concat('dbname=', data\_base\_name, ' options=-csearch\_path='),

'select conname, consrc

from pg\_constraint

where contype = ''c''

and (lower(consrc) like ''% like %'' or consrc like ''% ~~ %'')')

as t1(con\_name varchar, con\_src varchar)

loop

raise info 'Name: %, src: %', constraint\_rec.con\_name, constraint\_rec.con\_src;

end loop;

end

$$;

-- Тестируем

-- Добавили ограничение с like

alter table customers

add constraint a\_in\_name check ( name like '%a%');

-- Вызвали процедуру

DO

$$

begin

call get\_like\_constraints('rk2');

end;

$$;

----------------------------------------------------

-- Создать хранимую процедуру с входным параметром – "имя таблицы",

-- которая удаляет дубликаты записей из указанной таблицы в текущей

-- базе данных. Созданную процедуру протестировать.

----------------------------------------------------

create or replace procedure rem\_duplicates(in t\_name text)

language plpgsql

as

$$

declare

query text;

col text;

column\_names text[];

begin

query = 'delete from ' || t\_name || ' where id in (' ||

'select ' || t\_name || '.id ' ||

'from ' || t\_name ||

' join (select id, row\_number() over (partition by ';

for col in select column\_name from information\_schema.columns where information\_schema.columns.table\_name=t\_name loop

query = query || col || ',';

end loop;

query = trim(trailing ',' from query);

query = query || ') as rn from ' || t\_name || ') as t on t.id = ' || t\_name || '.id' ||

' where rn > 1)';

raise notice '%', query;

execute query;

end

$$;

-- Тестируем

-- Добавили дубликаты

insert into teacher(id, dep\_id, name, grade, job)

select \*

from teacher

where id < 5;

-- Вызвали процедуру

DO

$$

begin

call rem\_duplicates('teacher');

end;

$$;

----------------------------------------------------

﻿--Создать хранимую процедуру с входным параметром – имя таблицы,

--которая выводит сведения об индексах указанной таблицы в текущей базе

--данных. Созданную хранимую процедуру протестировать.

----------------------------------------------------

DROP PROCEDURE IF EXISTS getIdx;

CREATE PROCEDURE getIdx(n VARCHAR(30))

AS $$

DECLARE

a INT;

curRow RECORD;

tblCurs REFCURSOR;

BEGIN

OPEN tblCurs FOR

EXECUTE 'SELECT indexname FROM pg\_indexes WHERE tablename =' || n;

LOOP

FETCH tblCurs INTO curRow;

EXIT WHEN NOT FOUND;

RAISE NOTICE '%', curRow.indexname;

END LOOP;

CLOSE tblCurs;

END;

$$ LANGUAGE PLpgSQL;

CALL getIdx('Teacher');