

Университет ИТМО
Факультет программной инженерии и компьютерной техники

Лабораторная работа №1
по дисциплине «Распределенные системы хранения данных»

Выполнил:
Студент группы Р3331
Дворкин Борис Александрович
Вариант: 368090

Преподаватель:
Николаев Владимир Вячеславович

г. Санкт-Петербург

2024 г.

Содержание

Описание задания.....	3
Выполнение.....	3
Вывод.....	6

Описание задания

Введите вариант: 368090

Используя сведения из системных каталогов, получить информацию обо всех триггерах, назначенных на указанную таблицу схемы.

COLUMN NAME	TRIGGER NAME
-----	-----
COLUMN1	TRIGGER1
COLUMN2	TRIGGER2
...	

Программу оформить в виде анонимного блока.

Выполнение

Старый вариант, в котором я пользовался *variable interpolation*:

```
-- 1) Prompt the user for a table name
\prompt 'Enter the table name: ' my_table

DO $$
DECLARE
    in_table TEXT := 'my_table';
    found_table TEXT;
    rec RECORD;           -- для перебора триггеров
    v_attname TEXT;       -- имя столбца триггера
    v_arr_len INTEGER;    -- длина массива tgattr
BEGIN

-----

-- А. БАЗОВАЯ ВАЛИДАЦИЯ И ТЕКУЩИЕ ПАРАМЕТРЫ
-----

    IF in_table IS NULL OR in_table = '' THEN
        RAISE NOTICE 'Table name cannot be empty.';
    RETURN;
    END IF;
```

```

RAISE NOTICE 'Current database: %', current_database();
RAISE NOTICE 'Current schema:  %', current_schema();
RAISE NOTICE 'User input table: %', in_table;

-----
-- В. ПРОВЕРКА, ЧТО ТАБЛИЦА СУЩЕСТВУЕТ В ТЕКУЩЕЙ СХЕМЕ
-----

SELECT c.relname
INTO found_table
FROM pg_class c
      JOIN pg_namespace n ON n.oid = c.relnamespace
WHERE n.nspname = current_schema()
AND c.relkind = 'r'          -- r = обычная таблица
AND c.relname = in_table
LIMIT 1;

IF found_table IS NULL THEN
  RAISE NOTICE 'Table "%" does not exist in schema "%".',
in_table, current_schema();
RETURN;
END IF;

RAISE NOTICE 'Using table: %', found_table;

-----
-- С. ВЫВОД ЗАГОЛОВКА (COLUMN NAME, TRIGGER NAME)
-----

RAISE NOTICE 'COLUMN NAME          TRIGGER NAME';
RAISE NOTICE '-----';

-----
-- Д. ПЕРЕБОР ТРИГГЕРОВ ДЛЯ УКАЗАННОЙ ТАБЛИЦЫ
-----

FOR rec IN
  SELECT t.tgname,
         t.tgattr,
```

```

        c.oid AS table_oid
    FROM pg_trigger t
    JOIN pg_class c      ON t.tgrelid = c.oid
    JOIN pg_namespace ns ON c.relnamespace = ns.oid
WHERE ns.nspname      = current_schema()
    AND c.relname      = found_table
    AND NOT t.tgisinternal
LOOP
v_arr_len := array_length(rec.tgattr, 1);

IF v_arr_len IS NULL THEN
    -- Если tgattr пуст, триггер срабатывает на изменения
    -- любых столбцов
    RAISE NOTICE '%', 'ALL', rec.tgname;
ELSE
    -- Иначе выводим имена столбцов
    FOR i IN 1..v_arr_len LOOP
        SELECT attname
        INTO v_attname
        FROM pg_attribute
        WHERE attrelid = rec.table_oid
        AND attnum     = rec.tgattr[i];

        RAISE NOTICE '%', v_attname,
rec.tgname;
    END LOOP;
END IF;
END LOOP;

EXCEPTION WHEN SQLSTATE '42501' THEN
-----
-----
-- Е. ОБРАБОТКА ОШИБКИ НЕДОСТАТОЧНЫХ ПРАВ
-----
-----

    RAISE NOTICE 'Insufficient privileges to read system catalogs.
Check your user rights.';
END
$$ LANGUAGE plpgsql;

```

Новый вариант, добавил пару фиш:

- Передача переменной с помощью параметра, его установка с помощью SET (GUC-параметр), и обработка связанным с ним ошибок с помощью мета-команд постгреса

```
\if :{?my_table}
    SET "my.table" TO :'my_table';
\else
    \echo "ERROR: Set 'my_table' variable. Example: psql -v
my_table=schema.table"
    \quit
\endif
```

- Фиши в анонимном блоке:
 - current_setting() - встроенная функция постгреса, чтоб получить значение параметра конфигурации (GUC-параметра) текущей сессии. Таким образом удалось закомментировать передачу аргумента в анонимный блок, причём достаточно удобно, ибо вызов блока выглядит вот так:

```
psql -h pg -d studs -v my_table="s368090.test_table" -f script.sql
```

- split_part() - тоже встроенная функция постгреса, с помощью неё я обрабатываю всевозможные форматы, в которых пользователь может передать табличку.
- убрал огромные комментарии. Они только мешали 😊

```
DO $$
DECLARE
    in_table    TEXT := current_setting('my.table');
    input_schema TEXT;
    input_table TEXT;
    rec         RECORD;
    v_attnum    SMALLINT;
    v_attname   TEXT;
    output_text TEXT;
    col_array   SMALLINT[];
BEGIN
    IF position('.') IN in_table) > 0 THEN
```

```

input_schema := split_part(in_table, '.', 1);
input_table  := split_part(in_table, '.', 2);
ELSE
input_schema := current_schema();
input_table  := in_table;
END IF;

PERFORM 1
FROM pg_class c
JOIN pg_namespace n ON n.oid = c.relnamespace
WHERE n.nspname = input_schema
AND c.relname = input_table
AND c.relkind = 'r';

IF NOT FOUND THEN
RAISE EXCEPTION 'Table "%" not found!', in_table;
END IF;

RAISE NOTICE 'COLUMN NAME          TRIGGER NAME';
RAISE NOTICE '-----';
-----';

FOR rec IN
SELECT
    t.tgname,
    t.tgattr,
    c.oid AS table_oid,
    c.relname AS table_name
FROM pg_trigger t
JOIN pg_class c ON t.tgrelid = c.oid
JOIN pg_namespace n ON c.relnamespace = n.oid
WHERE
    n.nspname = input_schema
    AND c.relname = input_table
    AND NOT t.tgisinternal
LOOP
col_array := (
    SELECT array_agg(attnum::SMALLINT)
    FROM unnest(rec.tgattr::int2[]) AS attnum

```

```

        WHERE attnum IS NOT NULL
    );

    IF col_array IS NOT NULL AND array_length(col_array, 1) > 0
    THEN
        FOREACH v_attnum IN ARRAY col_array
        LOOP
            SELECT attname INTO v_attname
            FROM pg_attribute
            WHERE
                attrelid = rec.table_oid
                AND attnum = v_attnum;

            IF FOUND THEN
                output_text := v_attname || repeat(' ', 20 -
length(v_attname)) || rec.tgname;
                RAISE NOTICE '%', output_text;
            ELSE
                RAISE WARNING 'Column % not found in table %',
v_attnum, rec.table_name;
            END IF;
        END LOOP;
    ELSE
        output_text := 'ALL' || repeat(' ', 20) || rec.tgname;
        RAISE NOTICE '%', output_text;
    END IF;
END LOOP;

EXCEPTION
    WHEN SQLSTATE '42501' THEN
        RAISE NOTICE 'Insufficient privileges to read system catalogs
or table.';
    WHEN SQLSTATE '42P01' THEN
        RAISE NOTICE 'Table does not exist (SQLSTATE 42P01).';
    WHEN OTHERS THEN
        RAISE NOTICE 'Unexpected error: %', SQLERRM;
END
$$ LANGUAGE plpgsql;

```


Вывод

Кластер->бд->схема->объекты. Интересная лаба, вспомнить былое.

А теперь об интересном:

1. Проблема с **tgattr** и колоночными триггерами

Что такое **tgattr**:

Поле в системной таблице **pg_trigger** типа **int2vector**, хранящее номера столбцов (attnum), на которые действует триггер.

int2vector Struct Reference	
#include <c.h>	
Data Fields	
int32	vl_len_
int	ndim
int32	dataoffset
Oid	elemtype
int	dim1
int	lbound1
int16	values [FLEXIBLE_ARRAY_MEMBER]

Пример:

tgattr = {2} → триггер срабатывает при изменении столбца с номером 2.

Сложности:

- **int2vector** — это нестандартный массив, сложный для обработки.
- **tgattr** мог содержать **NULL** или некорректные значения после удаления столбцов.

Решение:

```
col_array := (  
    SELECT array_agg(attnum::SMALLINT)  
    FROM unnest(tgattr::int2[]) AS attnum  
    WHERE attnum IS NOT NULL  
);
```

Преобразование `int2vector` в обычный массив с фильтрацией `NULL`.

2. Обработка исключений прав доступа

Проблема:

При отсутствии прав на чтение системных таблиц (`pg_class`, `pg_trigger`) скрипт падал с ошибкой.

Решение:

Блок `EXCEPTION` с обработкой `SQLSTATE 42501`:

```
EXCEPTION WHEN SQLSTATE '42501' THEN  
    RAISE NOTICE 'Insufficient privileges to read system catalogs.';
```

Теперь при отсутствии прав выводится понятное сообщение.

3. Самая сложная проблема: Интерполяция переменных в анонимных блоках

Суть:

После обновления PostgreSQL перестала работать подстановка переменных `: 'my_var'` внутри `DO`-блоков:

```
DO $$  
BEGIN  
    RAISE NOTICE '%', : 'my_var'; -- Ошибка: ":" внутри строки!  
END  
$$;
```

Причина:

- Переменные `psql (:var)` интерполируются на клиенте до отправки SQL на сервер.
- В новых версиях парсер PostgreSQL стал строже проверять синтаксис внутри строковых литералов (`$$`), интерпретируя `:` как ошибку.

Решение:

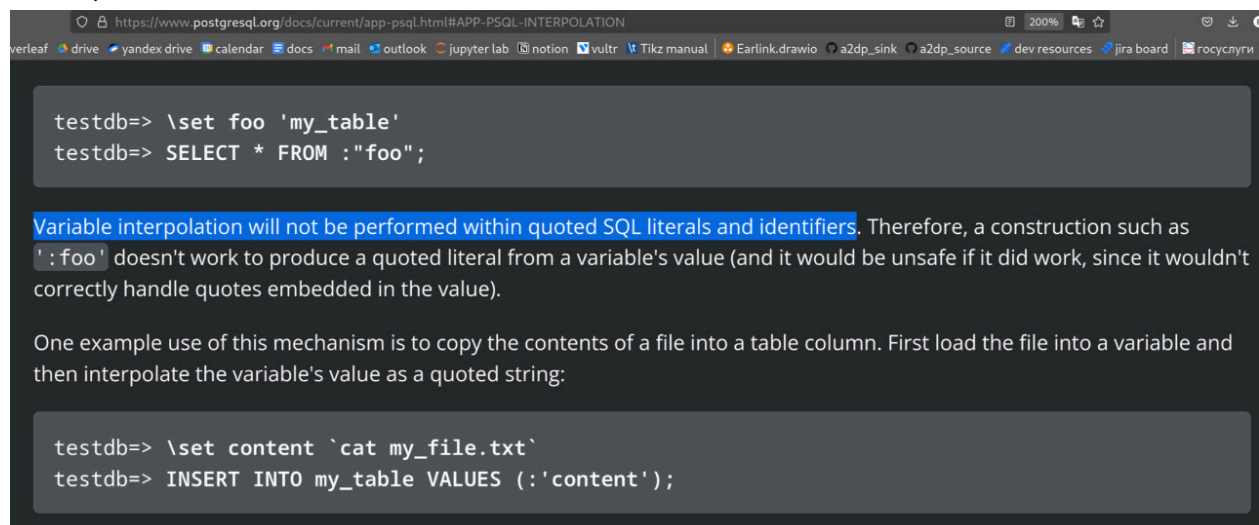
Использование GUC-параметров:

```
-- Установка параметра
SET "my.var" TO 'value';

-- Получение в блоке
DO $$
BEGIN
    RAISE NOTICE '%', current_setting('my.var');
END
$$;
```

P.S. Вот ссылки на source:

17 версия:



testdb=> \set foo 'my_table'

testdb=> SELECT * FROM :foo";

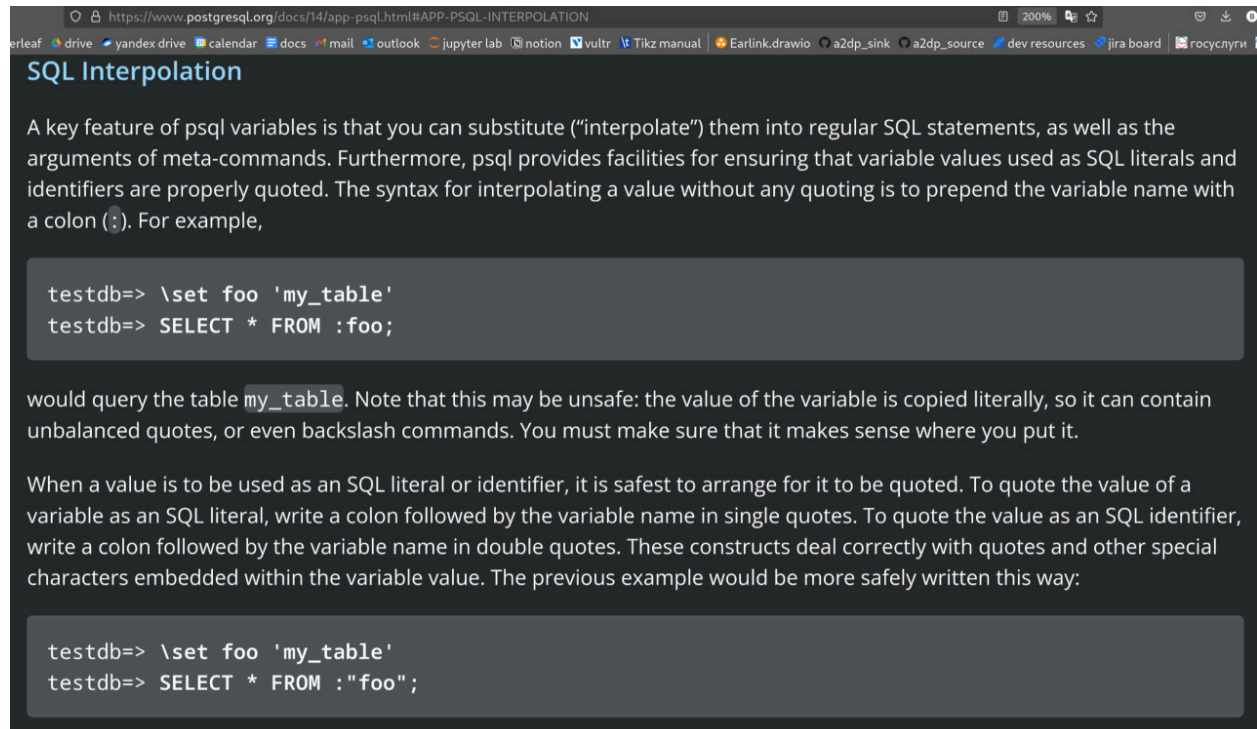
Variable interpolation will not be performed within quoted SQL literals and identifiers. Therefore, a construction such as `:foo` doesn't work to produce a quoted literal from a variable's value (and it would be unsafe if it did work, since it wouldn't correctly handle quotes embedded in the value).

One example use of this mechanism is to copy the contents of a file into a table column. First load the file into a variable and then interpolate the variable's value as a quoted string:

testdb=> \set content `cat my_file.txt`

testdb=> INSERT INTO my_table VALUES (:content');

14 версия:



The screenshot shows the PostgreSQL 14 documentation page for SQL Interpolation. The browser's address bar shows the URL: <https://www.postgresql.org/docs/14/app-psql.html#APP-PSQL-INTERPOLATION>. The page title is "SQL Interpolation". The text explains that a key feature of psql variables is the ability to substitute ("interpolate") them into regular SQL statements. It also mentions that psql provides facilities for ensuring that variable values used as SQL literals and identifiers are properly quoted. The syntax for interpolating a value without any quoting is to prepend the variable name with a colon (:). For example, the following commands are shown in a code block:

```
testdb=> \set foo 'my_table'
testdb=> SELECT * FROM :foo;
```

The text then states that this would query the table `my_table`. It notes that this may be unsafe because the value of the variable is copied literally, so it can contain unbalanced quotes, or even backslash commands. It advises that the value must make sense where it is used.

When a value is to be used as an SQL literal or identifier, it is safest to arrange for it to be quoted. To quote the value of a variable as an SQL literal, write a colon followed by the variable name in single quotes. To quote the value as an SQL identifier, write a colon followed by the variable name in double quotes. These constructs deal correctly with quotes and other special characters embedded within the variable value. The previous example would be more safely written this way:

```
testdb=> \set foo 'my_table'
testdb=> SELECT * FROM :'foo';
```

Как можно было заметить, в версии документации к 14му постгресу нету упоминания о том, что такой синтаксис запрещён. А всё потому, что он не запрещен, он работает 😊

У меня всё 🤞