Консоль.Функции. Триггеры.

Лекция 1.
Продвинутый SQL

нетология

Николай Хащанов

Full–stack developer

О спикере:

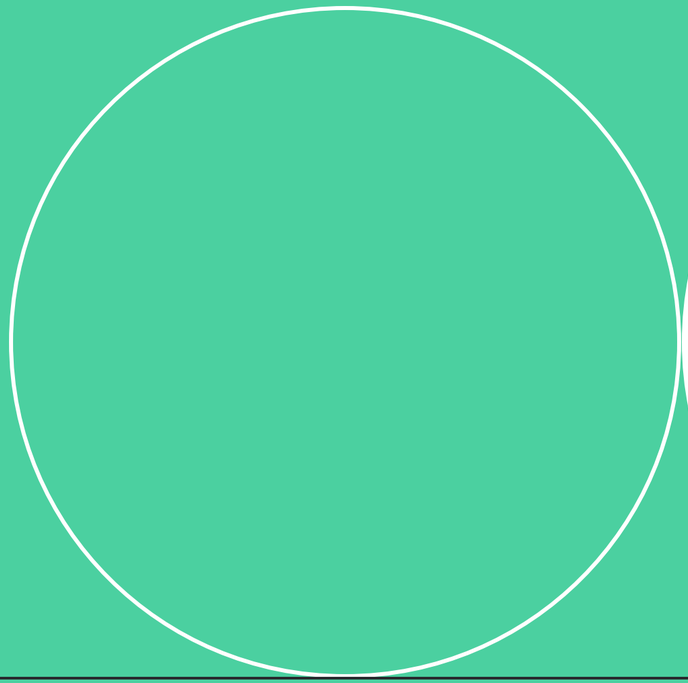
- Разрабатываю и поддерживаю crm/erp системы
- Преподаю в Нетологии
- Окончил РГТЭУ по специальности Менеджмент
- Оптимизация и автоматизация бизнес-процессов

Я в Слаке:

@Николай Хащанов

Содержание

Командная строка



1

Николай Хащанов

Лекция 1.
Консоль.Функции.Триггеры.

нетология

Командная строка

Командная строка используется для ввода и выполнения команд, запуска приложений и взаимодействия с операционной системой

Командная строка, а не интерфейс

1. Интерфейсы могут быть уязвимы с точки зрения безопасности
2. Размер импортируемого файла больше, чем поддерживает интерфейс
3. Любой интерфейс использует консольные команды
4. Отключение проверки работы ограничений

Как запустить?

В Windows:

1. Пуск -> В поиске набираете «cmd» -> выбираете «cmd.exe»
2. Сочетание клавиш Win + R -> набираете «cmd» -> Enter

В Mac OS:

1. Launchpad -> Другие -> Терминал
2. Spotlight -> набираете «Терминал» -> выбираете «Терминал»

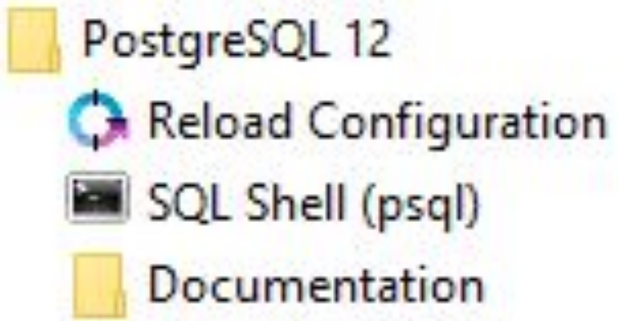
Командная строка и интерактивный режим

Работать с PostgreSQL можно через командную строку или интерактивный режим, представленный программой psql.

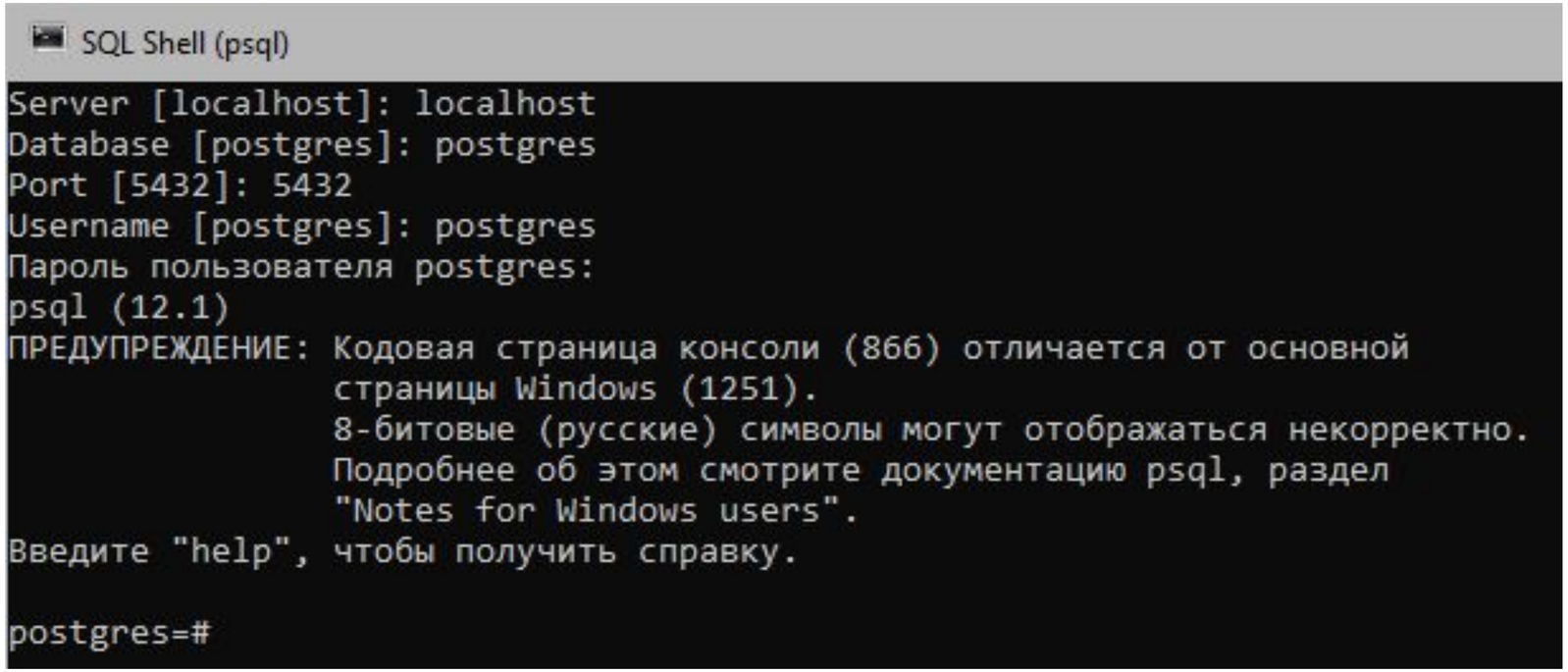
Рассмотрим интерактивный режим так как он универсален в плане используемой операционной системы.

Для работы потребуется установленный локально клиент PostgreSQL

https://www.enterprisedb.com/downloads/postgres-postgresql-downloads



Подключение



*Пароль при вводе не отображается

Кодировка в Windows:

psql – это «консольное приложение»

В Windows консольные окна используют кодировку символов отличную от той, что используется для остальной системы, нужно проявить особую осторожность при использовании 8-битных символов.

Если psql обнаружит проблемную кодовую страницу консоли, он предупредит вас при запуске. Чтобы изменить кодовую страницу консоли, необходимо:

1. Установите консольный шрифт в **Lucida Console**, потому что растровый шрифт не работает с кодовой страницей ANSI. В верхнем левом углу окна кликнуть на иконку, во всплывающем меню выбрать «Свойства», вкладка «Шрифт» и выбрать нужный.
2. Выполнить команду **\! chcp 1251**

3. Выполнить команду **set client_encoding='win1251';**

Практика

1. Давайте создадим базу данных: create database netology;

```
postgres=# create database netology;
CREATE DATABASE
```

2. Выведем список баз данных: \l

```
postgres=# \l
                                  Список баз данных
   Имя    | Владелец | Кодировка |     LC_COLLATE      |      LC_CTYPE       |     Права доступа
----------+----------+-----------+---------------------+---------------------+-----------------------
 netology | postgres | UTF8      | Russian_Russia.1251 | Russian_Russia.1251 |
 postgres | postgres | UTF8      | Russian_Russia.1251 | Russian_Russia.1251 |
 template0 | postgres | UTF8      | Russian_Russia.1251 | Russian_Russia.1251 | =c/postgres          +
          |          |           |                     |                     | postgres=CTc/postgres
 template1 | postgres | UTF8      | Russian_Russia.1251 | Russian_Russia.1251 | =c/postgres          +
          |          |           |                     |                     | postgres=CTc/postgres
(4 строки)
```

3. Выберем базу данных: \c netology;

```
postgres=# \c netology;
Вы подключены к базе данных "netology" как пользователь "postgres".
```

Практика

4. Создадим таблицу: create table students(id serial primary key, first_name varchar(50) not null, last_name varchar(50) not null);

```
netology=# create table students(id serial primary key,first_name varchar(50) not null, last_name varchar(50) not null);
CREATE TABLE
```

5. Посмотрим все таблицы в базе: \d

```
netology=# \d
                  Список отношений
 Схема  |       Имя        |        Тип         | Владелец
--------+------------------+--------------------+----------
 public | students         | таблица            | postgres
 public | students_id_seq  | последовательность | postgres
(2 строки)
```

6. Посмотрим содержимое определенной таблицы: \d students

```
netology=# \d students
                                  Таблица "public.students"
  Столбец   |          Тип           | Правило сортировки | Допустимость NULL |             По умолчанию
------------+------------------------+--------------------+-------------------+--------------------------------------
 id         | integer                |                    | not null          | nextval('students_id_seq'::regclass)
 first_name | character varying(50)  |                    | not null          |
 last_name  | character varying(50)  |                    | not null          |
Индексы:
    "students_pkey" PRIMARY KEY, btree (id)
```

Create, Read, Update, Delete:

Операции CRUD в командной строке имеют такой же синтаксис, как и в любом другом интерфейсе, они ни чем не отличаются

Команды psql:

\connect db_name – подключиться к базе с именем db_name

\du – список пользователей

\dp (или \z) – список таблиц, представлений, последовательностей, прав доступа к ним

\di – индексы

\ds – последовательности

\dt – список таблиц

\dt+ — список всех таблиц с описанием

\dv – представления

\dS – системные таблицы

\d+ – описание таблицы

\o – пересылка результатов запроса в файл

\l – список баз данных

\i – читать входящие данные из файла

\d “table_name” – описание таблицы

\i запуск команды из внешнего файла, например \i /my/directory/my.sql

\pset – команда настройки параметров форматирования

\echo – выводит сообщение

\set – устанавливает значение переменной среды. Без параметров выводит список текущих переменных (\unset – удаляет).

\? – справочник psql

\help – справочник SQL

\q (или Ctrl+D) – выход с программы

Команды командной строки:

-c (или –command) – запуск команды SQL без выхода в интерактивный режим

-f file.sql — выполнение команд из файла file.sql

-l (или –list) – выводит список доступных баз данных

-U (или –username) – указание имени пользователя

-W (или –password) – указание пароля пользователя

-d db_name — подключение к базе данных db_name

-h – имя хоста

-s – пошаговый режим, то есть, нужно будет подтверждать все команды

–S – однострочный режим, то есть, переход на новую строку будет выполнять запрос (избавляет от ; в конце конструкции SQL)

-V – версия PostgreSQL без входа в интерактивный режим

Командная строка:

Восстановление данных:

psql -h localhost -U user -d db_name -f my_db.sql

Резервная копия данных:

pg_dump -h localhost -p 5432 -U user -F c -b -v -f my_db.backup my_db

-h host - хост, если не указан то используется localhost или значение из переменной окружения PGHOST.

-p port - порт, если не указан то используется 5432 или значение из переменной окружения PGPORT.

-U - пользователь, если не указан то используется текущий пользователь, также значение можно указать в переменной окружения PGUSER.

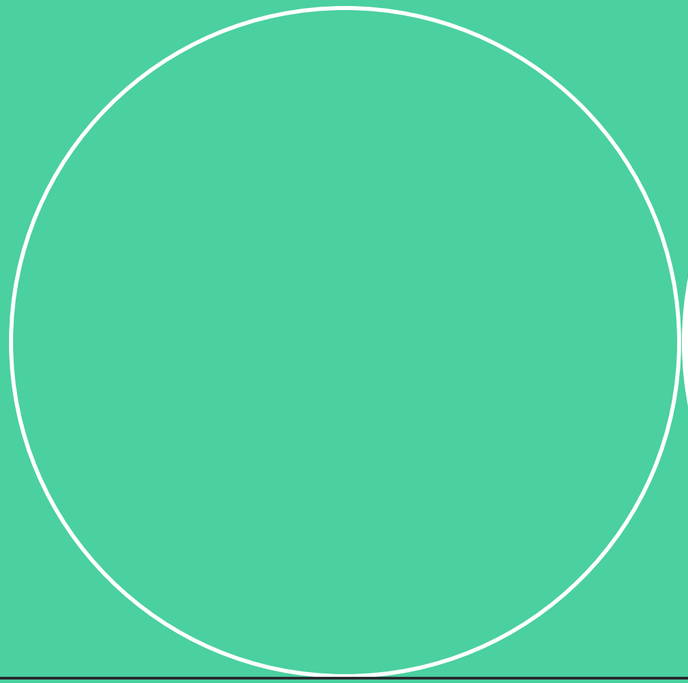
-F, --format {c|t|p} - выходной формат дампа, custom, tar, или plain text.

-b - включать в дамп большие объекты (blog’и).

-v, --verbose - вывод подробной информации.

-f file.sql - выполнение команд из файла file.sql

Функции



2

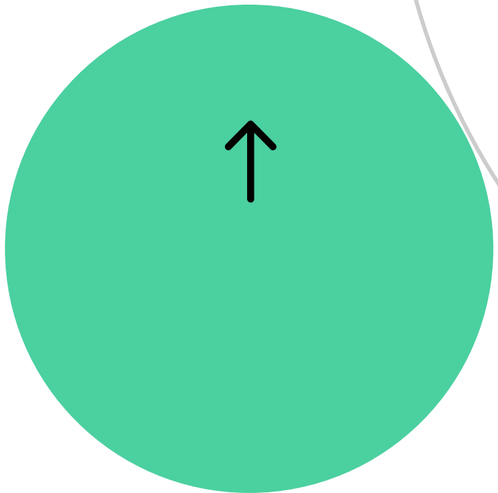
Николай Хащанов

Лекция 1.
Консоль.Функции.Триггеры.

нетология

Функции

применяются для осуществления манипуляций над данными и возврата результата в виде одного значения, множества или null



Функции. Для чего нужны?

1. Инкапсуляция. Внедрение логики в основной запрос.
2. Автоматизация процессов
3. Поддержка процедурного программирования
4. Написание собственных представлений с параметрами и аргументами

Виды функций

1. Функции на языке запросов (функции, написанные на SQL)
2. Функции на процедурных языках (например PL/pgSQL)
3. Внутренние функции
4. Функции на языке C

Преимущества PL/pgSQL:

1. Функции могут принимать и возвращать полиморфные типы anyelement, anyarray, anynonarray, anyenum и anyrange. В таких случаях фактические типы данных могут меняться от вызова к вызову

2. Функции могут возвращать «множества» (или таблицы) любого типа, которые могут быть возвращены в виде одного объекта. Такие функции генерируют вывод, выполняя команду RETURN NEXT для каждого элемента результирующего набора или RETURN QUERY для вывода результата запроса.

3. Функции могут возвращать void (отсутствие возвращаемого значения)

4. Функции можно объявить с выходными параметрами вместо явного задания типа возвращаемого значения. Это не добавляет никаких фундаментальных возможностей языку, но часто бывает удобно, особенно для возвращения нескольких значений. Нотация RETURNS TABLE может использоваться вместо RETURNS SETOF.

Структура PL/pgSQL:

Создание функции:

CREATE FUNCTION имя_функции(параметры)

RETURNS integer AS $$

'тело_функции’

$$ LANGUAGE plpgsql;

Текст тела функции должен быть блоком:

[<<метка>>]

[DECLARE

объявления]

BEGIN

операторы

END [метка];

Объявление параметров функции:

Переданные в функцию параметры именуются идентификаторами $1, $2 и т. д.

Неявное объявление:

```
CREATE FUNCTION имя_функции(имя_параметра_1 тип_данных, имя_параметра_2 тип_данных)
RETURNS тип_данных AS $$ ...
```

Явное объявление:

```
CREATE FUNCTION имя_ функции(тип_данных) RETURNS тип_данных AS $$
DECLARE
    имя_параметра ALIAS FOR $1; ...
```

Выходные параметры объявляются так же, только добавляется оператор OUT:

```
CREATE FUNCTION имя_функции(имя_параметра_1 тип_данных, OUT имя_параметра_2 тип_данных)
RETURNS тип_данных AS $$ ...
```

Переменные:

Объявление:

DECLARE имя_переменной [CONSTANT] тип_данных [NOT NULL] [{ DEFAULT |:= } значение]

Пример объявление простой переменной:

DECLARE age integer := 35;

* начальное значение переменной age - 35

Пример объявления константы:

DECLARE country CONSTANT varchar DEFAULT ‘Россия’;

*изменить константу country после объявления будет невозможно.

Практика 1

Давайте напишем функцию, которая будет возводить число в степень

```
CREATE FUNCTION power_x_y(x int, y int, OUT x_in_y int)
AS $$
BEGIN
    x_in_y := power(x, y);
END;
$$ LANGUAGE plpgsql;

select power_x_y(2, 2) --4

select power_x_y(3, 5) -- 243
```

Нет ничего сложного!

Практика 2

Напишем функцию, которая будет считать количество записей в каждой таблице знакомой нам базы dvd-rental:

```
CREATE OR REPLACE FUNCTION some_func() RETURNS SETOF text AS $$
DECLARE
    t record;
    c integer;
BEGIN
    FOR t IN
        SELECT table_name FROM information_schema.TABLES
        WHERE table_schema = 'dvd-rental' AND table_type != 'VIEW'
        ORDER BY table_name DESC
    LOOP
        EXECUTE 'select count(*) cnt FROM ' || t.table_name INTO c;
        return next t::text || ' - ' || c::text;
    END LOOP;
END $$
LANGUAGE plpgsql;

select some_func()

drop function some_func()
```

	some_func
1	(store) - 2
2	(staff) - 2
3	(rental) - 16044
4	(payment) - 14596
5	(orders) - 4
6	(language) - 6
7	(inventory) - 4581
8	(film_category) - 1000
9	(film_actor) - 5462
10	(film) - 1000
11	(customer) - 599
12	(country) - 109
13	(city) - 600
14	(category) - 16
15	(author) - 3
16	(address) - 603
17	(actor) - 200

Ключевые моменты практики 2

- При создании функции можно использовать ключевое слово REPLACE
- SETOF позволяет возвращать множество значений, для этого в каждом цикле используем RETURN NEXT
- В конструкции FOR .. IN если используем запрос, то *цель – t* имеет тип RECORD, в противном случает тип будет INTEGER
- Оператор EXECUTE позволяет выполнять запросы внутри функции
- Оператор INTO позволяет записать результат запроса в переменную
- Для удаления функции используем конструкцию **drop function имя_функции**

О чем стоит помнить

- При добавлении в текст запроса переменной, содержащей идентификаторы столбцов и таблиц, нужно использовать функцию quote_ident
- Если переменная может быть пустой, желательно использовать функцию quote_nullable
- В PL/pgSQL вместо совместимого с plSQL := можно использовать просто =
- При использовании INTO в переменную записывается первая строка, возвращенная запросом, для упорядочивания результата в запросе желательно использовать ORDER BY
- Вместо RETURNS SETOF можно использовать RETURNS TABLE(выходные параметры), но для этого выходные параметры нужно объявить
- Для указания переменных в конструкции EXECUTE можно использовать USING. EXECUTE 'SELECT ... FROM ... WHERE id = $1’ INTO ... USING имя_переменной;

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

две формы CASE:

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

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

```
create function delete_not_active(id int) returns text as $$
declare
    a int;
    b text;
begin
    execute 'select active, last_name from customer where customer_id = ' || id into a, b;
    if
        a = 0
    then
        delete from customer where customer_id = id;
        return b || ' was deleted';
    else
        return b || ' can`t be deleted';
    end if;
end;
$$ language plpgsql;

select delete_not_active(17)
```

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

```
create function find_category(c text) returns text as $$
begin
    case c
        when
            'Action', 'Animation', 'Children', 'Classics', 'Comedy'
        then
            return 'yes';
        else
            return 'no';
        end case;
end;
$$ language plpgsql;

select find_category('Animation')
```

Простые циклы 1

- LOOP

  [<<метка>>] LOOP операторы END LOOP [метка];

- WHILE

  [<<метка>>] WHILE логическое-выражение LOOP операторы END LOOP [метка];

- FOREACH

  [<<метка>>] FOREACH цель [SLICE число] IN ARRAY выражение

  LOOP операторы END LOOP [метка];

    * SLICE указывается, если нужно работать со срезом массива

- EXIT

  EXIT [метка] [WHEN логическое-выражение];

- CONTINUE

  CONTINUE [метка] [WHEN логическое-выражение];

    * EXIT и CONTINUE можно использовать со всеми типами циклов

Простые циклы 2

- FOR (целочисленный)

  [<<метка>>] FOR имя IN [REVERSE] выражение .. выражение [BY выражение]

  LOOP операторы END LOOP [метка];

  * Переменная *имя* автоматически определяется с типом integer и существует только внутри цикла

- FOR (по результатам запроса)

  [<<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.

LOOP, FOR (целочисленный)

```
create function one_or_two() returns int as $$
declare
    a int = 0;
    c int;
begin
    execute 'select count(*) from customer' into c;
    for i in reverse c .. 1 by 2 loop
        execute 'select count(*) from rental where customer_id = ' || i into c;
        a = a + c;
    end loop;
    return a;
end;
$$ language plpgsql;

select one_or_two()
```

А как же insert, update, delete?

Все точно так же!

В функции можно произвести вычисления и результат записать в таблицу или выполнить обновление данных. Если запись не удовлетворяет условию в функции, то ее можно удалить.

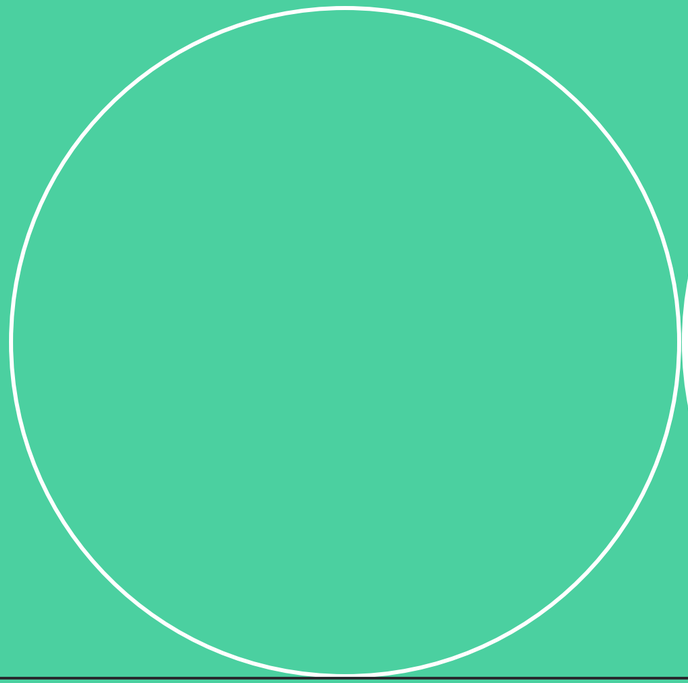
Важно помнить про конструкцию:

```
INSERT ... ON CONFLICT ... DO UPDATE (DO NOTHING)
```

Для обработки ошибок при изменении данных используется блок EXCEPTION:

```
BEGIN
    операторы
EXCEPTION WHEN условие [ OR условие ... ] THEN
    операторы_обработчика
END;
```

Триггеры



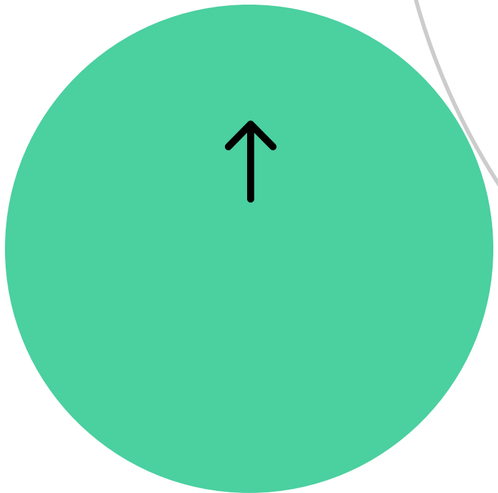
3

Николай Хащанов

Лекция 1.
Консоль.Функции.Триггеры.

нетология

Триггеры.
Зачем делать руками то, что можно автоматизировать?



Триггеры при изменении данных

Объявляется как функция без аргументов и с типом результата trigger.

Как часто возникает ситуация, что при изменении значения нужно проставить дату внесения изменений?

```
create function update_last_rental_date() returns trigger as $$
    begin
        new.last_update = now();
        return new;
    end;
$$ language plpgsql

create trigger update_rental_date
before update on rental
for each row execute function update_last_rental_date();
```

Специальные переменные 1

Когда функция на PL/pgSQL срабатывает как триггер, в блоке верхнего уровня автоматически создаются несколько специальных переменных:

NEW - Тип данных RECORD. Переменная содержит новую строку базы данных для команд INSERT/UPDATE в триггерах уровня строки. В триггерах уровня оператора и для команды DELETE эта переменная имеет значение null.

OLD - Тип данных RECORD. Переменная содержит старую строку базы данных для команд UPDATE/DELETE в триггерах уровня строки. В триггерах уровня оператора и для команды INSERT эта переменная имеет значение null.

TG_NAME - Тип данных name. Переменная содержит имя сработавшего триггера.

TG_WHEN - Тип данных text. Строка, содержащая BEFORE, AFTER или INSTEAD OF, в зависимости от определения триггера.

TG_LEVEL - Тип данных text. Строка, содержащая ROW или STATEMENT, в зависимости от определения триггера.

TG_OP - Тип данных text. Строка, содержащая INSERT, UPDATE, DELETE или TRUNCATE, в зависимости от того, для какой операции сработал триггер.

Специальные переменные 2

TG_RELID - Тип данных oid. OID таблицы, для которой сработал триггер.

TG_RELNAME - Тип данных name. Имя таблицы, для которой сработал триггер. Эта переменная устарела и может стать недоступной в будущих релизах. Вместо неё нужно использовать TG_TABLE_NAME.

TG_TABLE_NAME - Тип данных name. Имя таблицы, для которой сработал триггер.

TG_TABLE_SCHEMA - Тип данных name. Имя схемы, содержащей таблицу, для которой сработал триггер.

TG_NARGS - Тип данных integer. Число аргументов в команде CREATE TRIGGER, которые передаются в триггерную функцию.

TG_ARGV[] - Тип данных массив text. Аргументы от оператора CREATE TRIGGER. Индекс массива начинается с 0. Для недопустимых значений индекса (< 0 или >= tg_nargs) возвращается NULL.

Как использовать специальные переменные:

```
create function process_table_audit() returns trigger as $$
    begin
        if (TG_OP = 'DELETE') then
            insert into emp_audit select 'D', now(), user, old.*, TG_TABLE_NAME;
        elsif (TG_OP = 'UPDATE') then
            insert into emp_audit select 'U', now(), user, new.*, TG_TABLE_NAME;
        elsif (TG_OP = 'INSERT') then
            insert into emp_audit select 'I', now(), user, new.*, TG_TABLE_NAME;
        end if;
        return null; -- возвращаемое значение для триггера AFTER игнорируется
    end;
$$ language plpgsql;

create trigger table_audit
after insert or update or delete on some_table
    for each row execute function process_table_audit();
```

Триггеры событий:

Объявляется как функция без аргументов и с типом результата event_trigger.

```
create or replace function alert() returns event_trigger as $$
begin
    raise notice 'alert : % %', tg_event, tg_tag;
end;
$$ language plpgsql;

create event trigger alert on update_last_rental_date execute function alert();
```

* Данный триггер выведет сообщение о срабатывании функции изменения даты в таблице rental.

Специальные переменные:

Когда функция на PL/pgSQL вызывается как событийный триггер, в блоке верхнего уровня автоматически создаются несколько специальных переменных:

TG_EVENT

Тип данных text. Строка, содержащая событие, для которого сработал триггер.

TG_TAG

Тип данных text. Переменная, содержащая тег команды, для которой сработал триггер.

Видимость изменений в данных

1

Триггеры уровня оператора следуют **простым правилам видимости**: никакие из изменений, произведённых оператором, не видны в триггерах BEFORE, тогда как в триггерах AFTER видны все изменения.

Видимость изменений в данных

1

Триггеры уровня оператора следуют простым правилам видимости: никакие из изменений, произведённых оператором, не видны в триггерах BEFORE, тогда как в триггерах AFTER видны все изменения.

2

Изменение данных (вставка, обновление или удаление), заставляющее сработать триггер, **не видно для команд SQL**, выполняемых в триггере BEFORE уровня строки, потому что это изменение ещё не произошло.

Видимость изменений в данных

1

Триггеры уровня оператора следуют простым правилам видимости: никакие из изменений, произведённых оператором, не видны в триггерах BEFORE, тогда как в триггерах AFTER видны все изменения.

2

Изменение данных (вставка, обновление или удаление), заставляющее сработать триггер, не видно для команд SQL, выполняемых в триггере BEFORE уровня строки, потому что это изменение ещё не произошло.

3

Тем не менее, команды SQL, выполняемые в триггере BEFORE уровня строки, **будут видеть изменения данных в строках**, которые уже были обработаны в этом операторе. Это требует осторожности, так как порядок обработки строк в целом непредсказуемый; команда SQL, обрабатывающая множество строк, может делать это в любом порядке.

Видимость изменений в данных

1

Триггеры уровня оператора следуют простым правилам видимости: никакие из изменений, произведённых оператором, не видны в триггерах BEFORE, тогда как в триггерах AFTER видны все изменения.

2

Изменение данных (вставка, обновление или удаление), заставляющее сработать триггер, не видно для команд SQL, выполняемых в триггере BEFORE уровня строки, потому что это изменение ещё не произошло.

3

Тем не менее, команды SQL, выполняемые в триггере BEFORE уровня строки, будут видеть изменения данных в строках, которые уже были обработаны в этом операторе. Это требует осторожности, так как порядок обработки строк в целом непредсказуемый; команда SQL, обрабатывающая множество строк, может делать это в любом порядке.

4

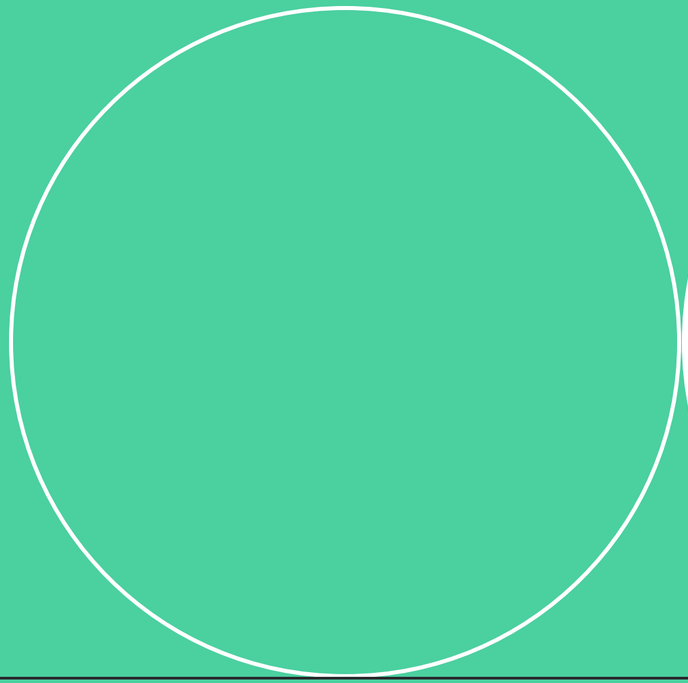
- Аналогично, триггер INSTEAD OF уровня строки **увидит изменения данных**, внесённые при предыдущих вызовах триггера INSTEAD OF для этой же внешней команды.
- Когда срабатывает триггер AFTER уровня строки, все изменения сделанные оператором **уже выполнены и видны** в вызываемой триггерной функции.

Важно помнить

Если есть несколько триггеров на одно и то же событие для одной и той же таблицы, то они будут вызываться в алфавитном порядке по имени триггера.

Для триггеров BEFORE и INSTEAD OF потенциально изменённая строка, возвращаемая одним триггером, становится входящей строкой для следующего триггера. Если любой из триггеров BEFORE или INSTEAD OF возвращает NULL, операция для этой строки прекращается и последующие триггеры (для этой строки) не срабатывают.

Итоги



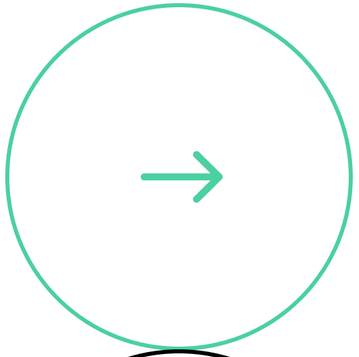
4

Николай Хащанов

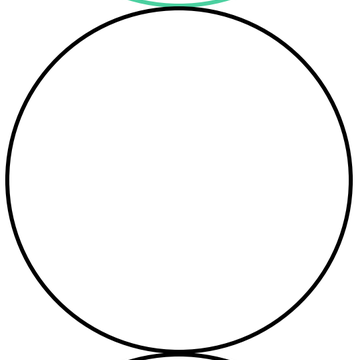
Лекция 1.
Консоль.Функции.Триггеры.

нетология

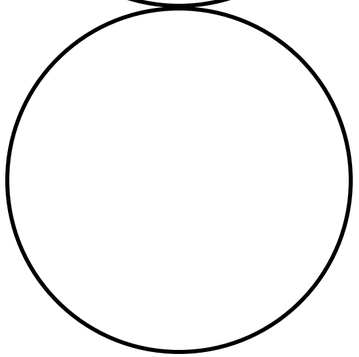
Подведем итог



Консоль, работа без GUI

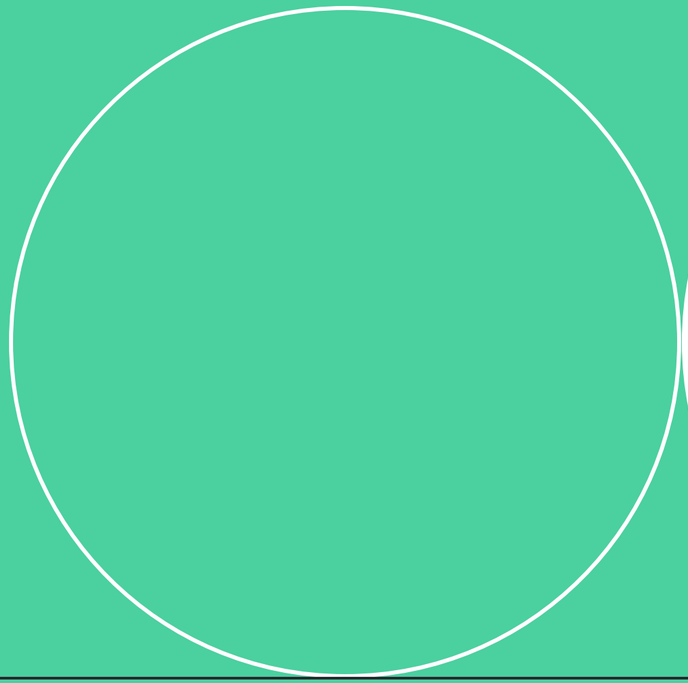


Функции – научились создавать, вычислять, работать с данными



Триггеры – автоматизируем процессы

Домашнее задание



5

Николай Хащанов

Лекция 1.
Консоль.Функции.Триггеры.

нетология

Задание 1. World-db.

- В командной строке используя pg_restore.exe восстановите бэкап базы world-db
- Выполните запрос *select * from country*
- Ссылка на бэкап: https://drive.google.com/file/d/193k6Aioj1P25YEt6sCMgIbre9RM7BHEg/view?usp=sharing

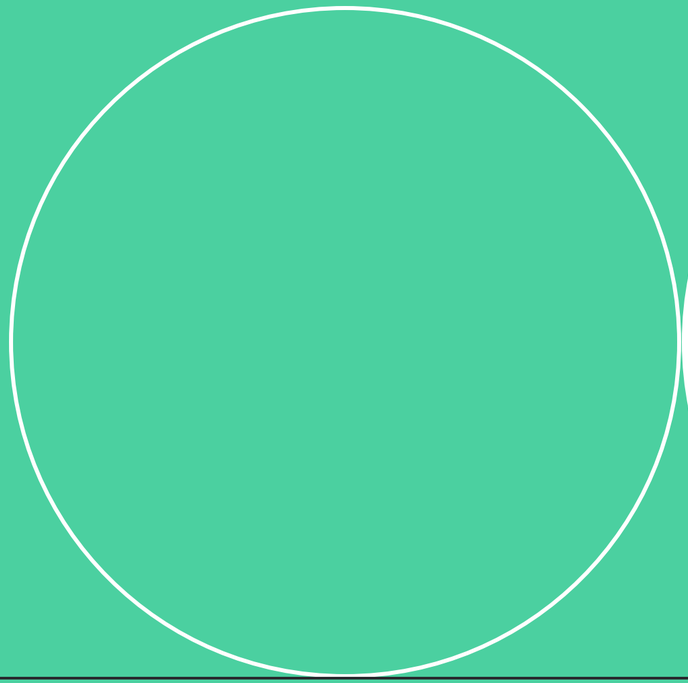
Задание 2. Dvd-rental.

- Напишите функцию, которая будет в виде аргументов принимать две даты и в качестве результата возвращать сумму продаж между этими датами, включая эти даты.

Задание 3. Dvd-rental.

- Создайте таблицу *not_active_customer* со столбцами *id, customer_id и not_active_date (дата создания записи)*
- Напишите триггерную функцию, которая будет срабатывать при изменении данных в таблице *customer*, если пользователь становится неактивным, то в таблицу *not_active_customer* должна добавиться запись об этом пользователе

Полезные ссылки



6

Николай Хащанов

Лекция 1.
Консоль.Функции.Триггеры.

нетология

Интерактивный терминал и командная строка:

https://postgrespro.ru/docs/postgresql/12/app-psql

Восстановление и бэкап:

https://postgrespro.ru/docs/postgresql/12/backup

Процедурный язык PL/pgSQL:

https://postgrespro.ru/docs/postgresql/12/plpgsql

Спасибо за внимание!

Николай Хащанов

нетология