Лекция 4

Функции
Триггеры

Зачем нужны функции?

- иногда хочется сделать сложную логику на уровне СУБД
- выбрать 1кк записей, сделать над ними какие-то действия и записать обратно
- можно не пересылать данные в приложение и обратно, а просто сделать функцию

Синтаксис функций

```
CREATE [ OR REPLACE ] FUNCTION
    имя ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента [ { DEFAULT | = } выражение_по_умолчанию ] [, ...] ] )
    [ RETURNS тип_результата
      | RETURNS TABLE ( имя_столбца тип_столбца [, ...] ) ]
  { LANGUAGE имя_языка
    | TRANSFORM { FOR TYPE имя_типа } [, ... ]
    | WINDOW
    | IMMUTABLE | STABLE | VOLATILE | [ NOT ] LEAKPROOF
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | PARALLEL { UNSAFE | RESTRICTED | SAFE }
    | COST стоимость_выполнения
    | ROWS строк_в_результате
    | SUPPORT вспомогательная_функция
    | SET параметр_конфигурации { TO значение | = значение | FROM CURRENT }
    | AS 'определение'
    | AS 'объектный_файл', 'объектный_символ'
  } ...
```

https://postgrespro.ru/docs/postgresql/12/sql-createfunction

То же самое, но проще

```
create [or replace] function function_name(param_list)
   returns return_type
   language plpgsql
  as
$$
declare
-- variable declaration
begin
 -- logic
end;
$$
```

- функция должна содержать возвращаемое значение
- функции можно создавать на: SQL, PL/pgSQL, C, Python и не только

Синтаксис процедуры

```
CREATE [ OR REPLACE ] PROCEDURE
    имя ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента [ { DEFAULT | = } выражение_по_умолчанию ] [, ...] ] )
  { LANGUAGE имя_языка
    | TRANSFORM { FOR TYPE имя_типа } [, ... ]
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | SET параметр_конфигурации { TO значение | = значение | FROM CURRENT }
    | AS 'определение'
    | AS 'объектный_файл', 'объектный_символ'
  } ...
```

https://postgrespro.ru/docs/postgresql/12/sql-createprocedure

Синтаксис процедуры

```
create [or replace] procedure procedure_name(parameter_list)
language plpgsql
as $$
declare
-- variable declaration
begin
-- stored procedure body
end; $$
```

Вызывается с помощью:

call stored_procedure_name(argument_list);

Разница между процедурой и функцией

- функция возвращает значение
- функции можно использовать в запросах
- хранимые процедуры могут начинать транзакции
- в PostgreSQL хранимые процедуры доступны с версии PostgreSQL 11.
- в более старых версиях PostgreSQL существуют только функции

Плюсы:

- не нужно тратить время на передачу данных к пользователю и обратно (сокращение трафика)
- кеширование плана запроса, сам запрос выполняется быстрее (оптимизация происходит на уровне БД)
- можно дать доступ только к хранимым процедурам, а не самим данным (из соображений безопасности)
- легче повторно использовать одинаковый код в разных приложениях

Минусы:

- трудности поддержания версий кода: процедуры хранятся в БД, сложно посмотреть предыдущую версию, в отличие от обычного кода
- тяжело проводить отладку
- миграция с одной СУБД на другую может привести к проблемам (в каждой СУБД свой синтаксис)

Пример

```
DROP FUNCTION IF EXISTS add(integer, integer);
CREATE FUNCTION add(integer, integer) RETURNS integer
    AS 'select $1 + $2;'
    LANGUAGE SQL
    IMMUTABLE
    RETURNS NULL ON NULL INPUT;

CREATE OR REPLACE FUNCTION inc(i integer) RETURNS integer AS $$
    BEGIN
            RETURN i + 1;
    END;
$$ LANGUAGE plpgsql;

SELECT inc(42), add(2, 3);
```

Доступ к переменным

- add(integer, integer)

  можно получать доступ к переменным как $1 и $2

- add(var1 integer, var2 integer)

  сделать переменные именованными и получать значение через имя

- А можно в DECLARE написать

  val ALIAS FOR $1;

```
CREATE FUNCTION func() RETURNS integer
    LANGUAGE plpgsql AS $$
    declare
    begin
      -- do something
    end
$$;
```

```
CREATE FUNCTION func() RETURNS INTEGER AS $$
    declare
    begin
      -- do something
    end
$$ LANGUAGE plpgsql;
```

Нет разницы писать объявление языка до или после функции
оба варианта допустимы

Dollar quoting $$

- $$ - это просто другой вариант написания кавычек в postgresql
- обычно кавычки экранируются ещё одними кавычками

'user's log' – так нельзя, кавычки не экранированы

'user''s log' – так можно

$$user's log$$ - то же самое

Dollar quoting $$

- Общий вид строковых констант

  $имя_тега$строка$имя_тега$
- когда нет имени тега = $$
- т.е. функции допустимо писать в одинарных кавычках ('), в $$ или в $your_tag_name$

Ещё раз коротко

- Создать функцию

  CREATE FUNCTION func_name ..
- Создать или заменить

  CREATE OR REPLACE FUNCTION func_name ..
- Удалить функцию

  DROP FUNCTION func_name;

PL/pgSQL

- содержит конструкции для определения порядка выполнения: условия, циклы и т.д.

PL/pgSQL: IF

```
IF boolean-expression THEN
    statements
[ ELSIF boolean-expression THEN
    statements
    ...]]
[ ELSE
    statements ]
END IF;
```

PL/pgSQL: LOOP

```
LOOP
    -- some computations
    EXIT WHEN count > 100;
    CONTINUE WHEN count < 50;
    -- some computations for count IN [50 .. 100]
END LOOP;
```

```
WHILE amount_owed > 0 LOOP
    -- some computations here
END LOOP;
```

```
WHILE NOT done LOOP
    -- some computations here
END LOOP;
```

PL/pgSQL: FOR

```
FOR i IN 1..10 LOOP
    -- i will take on the values 1,2,3,4,5,6,7,8,9,10 within the loop
END LOOP;
```

```
FOR i IN REVERSE 10..1 LOOP
    -- i will take on the values 10,9,8,7,6,5,4,3,2,1 within the loop
END LOOP;
```

```
FOR i IN REVERSE 10..1 BY 2 LOOP
    -- i will take on the values 10,8,6,4,2 within the loop
END LOOP;
```

PL/pgSQL: FOR target IN query LOOP

```
CREATE FUNCTION cs_refresh_mviews() RETURNS integer AS $$
DECLARE
    mviews RECORD;
BEGIN
    RAISE NOTICE 'Refreshing materialized views...';

    FOR mviews IN SELECT * FROM cs_materialized_views ORDER BY sort_key LOOP

        -- Now "mviews" has one record from cs_materialized_views

        RAISE NOTICE 'Refreshing view %s ...', quote_ident(mviews.mv_name);
        EXECUTE format('TRUNCATE TABLE %I', mviews.mv_name);
        EXECUTE format('INSERT INTO %I %s', mviews.mv_name, mviews.mv_query);
    END LOOP;

    RAISE NOTICE 'Done refreshing materialized views.';
    RETURN 1;
END;
$$ LANGUAGE plpgsql;
```

Изучаем, что ещё есть

- record - переменная типа - строка в таблице
- RAISE NOTICE

  RAISE - команда для вывода сообщений. NOTICE - уровень приоритета сообщения (в зависимости от уровня они могут только записываться в лог или будут переданы клиенту)

RAISE NOTICE

Возможные уровни: DEBUG, LOG, INFO, NOTICE, WARNING и EXCEPTION

EXCEPTION по умолчанию вызывает ошибку и прерывает транзакцию

https://postgrespro.ru/docs/postgresql/12/plpgsql-errors-and-messages

Строковые функции

- quote_ident(string text) - переданная строка оформляется для использования в качестве идентификатора в SQL-операторе
- format(formatstr text [, formatarg "any" [, ...]]) - форматирование строки

Строковые функции: https://postgrespro.ru/docs/postgrespro/12/functions-string

```
CREATE OR REPLACE FUNCTION test_if(numeric)
        RETURNS text AS
$BODY$
    DECLARE
       val ALIAS FOR $1;
       val1 integer;
       val2 text;
    BEGIN
        val1 :=1;
        IF val1 = val THEN
           val2 := 'Он работает';
        ELSE
           val2 := 'Не работает';
        END IF;
        RETURN val2;
    END;
$BODY$
    LANGUAGE 'plpgsql';
```

```
create function get_film_count(len_from int, len_to int)
returns int
language plpgsql
as
$$
declare
   film_count integer;
begin
   select count(*)
   into film_count
   from film
   where length between len_from and len_to;

   return film_count;
end;
$$;
```

into film_count - выбрать в переменную

Варианты вызова

```
select get_film_count(40,90);
```

```
select get_film_count(
    len_from => 40,
     len_to => 90
);
```

```
-- старый синтаксис (но поддерживается)
select get_film_count(
    len_from := 40,
    len_to := 90
);
```

```
select get_film_count(40, len_to => 90);
```

```
-- а вот так нельзя
select get_film_count(len_from => 40, 90);
```

Триггеры

- хранимые процедуры выполняются, когда их явно вызвали
- часто бывает необходимость вычислять эти хранимые процедуры не в момент явного вызова, а в момент изменения данных
- например, при изменении данных писать лог
- для этого существуют триггеры

Синтаксис создания триггера

```
CREATE [ CONSTRAINT ] TRIGGER имя { BEFORE | AFTER | INSTEAD OF } { событие [ OR ... ] }
    ON имя_таблицы
    [ FROM ссылающаяся_таблица ]
    [ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY DEFERRED ] ]
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( условие ) ]
    EXECUTE PROCEDURE имя_функции ( аргументы )

Здесь допускается событие:

    INSERT
    UPDATE [ OF имя_столбца [, ... ] ]
    DELETE
    TRUNCATE
```

Триггеры: переменные

- NEW: тип данных RECORD. Переменная содержит новую строку базы данных для команд INSERT/UPDATE.
- OLD: тип данных RECORD. Переменная содержит старую строку базы данных для команд UPDATE/DELETE.

Триггеры: переменные

- TG_NAME: тип данных name. Переменная содержит имя сработавшего триггера.
- TG_OP: тип данных text. Строка, содержащая INSERT, UPDATE, DELETE или TRUNCATE, в зависимости от того, для какой операции сработал триггер.
- TG_TABLE_NAME: тип данных name. Имя таблицы, для которой сработал триггер.

Даны таблицы

```
-- Таблица пользователей
CREATE TABLE users
(
  "name" text
)
```

```
-- Таблица логов
CREATE TABLE logs
(
  "text" text,
  "added" timestamp without time zone
)
```

Создание триггера

```
– создание триггера
CREATE TRIGGER t_user
AFTER INSERT OR UPDATE OR DELETE ON users FOR EACH ROW
EXECUTE PROCEDURE add_to_log ();

– удаление триггера
DROP TRIGGER IF EXISTS t_user ON users;
```

- t_user – имя триггера
- users – название таблицы
- add_to_log – имя функции
- триггер создаётся на добавление, удаление и изменение данных
- можно задавать только на добавление, например, или удаление и добавление (без обновления)

пока что нет самой функции add_to_log(), поэтому триггер работать не будет

```
CREATE OR REPLACE FUNCTION add_to_log() RETURNS TRIGGER AS $$
DECLARE
    mstr varchar(30);
    astr varchar(100);
    retstr varchar(254);
BEGIN
    IF    TG_OP = 'INSERT' THEN
        astr = NEW.name;
        mstr := 'Add new user ';
        retstr := mstr || astr;
        INSERT INTO logs(text,added) values (retstr,NOW());
        RETURN NEW;
    ELSIF TG_OP = 'UPDATE' THEN
        astr = NEW.name;
        mstr := 'Update user ';
        retstr := mstr || astr;
        INSERT INTO logs(text,added) values (retstr,NOW());
        RETURN NEW;
    ELSIF TG_OP = 'DELETE' THEN
        astr = OLD.name;
        mstr := 'Remove user ';
        retstr := mstr || astr;
        INSERT INTO logs(text,added) values (retstr,NOW());
        RETURN OLD;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

- новая функция, без входящих параметров, возвращает специальный тип TRIGGER
- в случае INSERT переменная NEW будет содержать новую строку, а OLD будет пустая
- в случае UPDATE обе переменные будут определены (соответствующими данными)
- в случае DELETE переменная NEW будет пустая, OLD содержать удаляемую строку

Когда срабатывать?

- триггер может срабатывать до события, после или вместо

  BEFORE | AFTER | INSTEAD OF

- событий может быть несколько

  AFTER INSERT OR UPDATE