Базы данных

Лекция 8.

Расширение SQL - PL/pgSQL. Функции, процедуры, триггеры.

МФТИ, 2024

Игорь Шевченко

@igorshvch

I. PL/pgSQL. Функции и процедуры

PL/pgSQL

- PL/pgSQL это процедурный язык для СУБД PostgreSQL
- PL/pgSQL:
  - используется для создания функций, процедур и триггеров
  - добавляет управляющие структуры к языку SQL
  - может выполнять сложные вычисления
- Функции PL/pgSQL могут использоваться везде, где допустимы встроенные функции
- Технологически PL/pgSQL – это загружаемый по умолчанию модуль, который может быть удален из компоновки конкретной СУБД администратором
- PL/pgSQL позволяет сгруппировать блок вычислений и последовательность запросов ***внутри*** сервера базы данных

PL/pgSQL

- Виды хранимых процедур:
  - Функции
    - Возвращают результат
    - Через синтаксис функций определяются триггеры
    - Функции PL/pgSQL используются в выражениях с операторами SQL так же, как и встроенные функции
    - В теле функции нельзя использовать операторы управления
    - транзакциями
  - Процедуры
    - Не возвращают результат
    - Для вызова необходимо использовать оператор CALL
    - могут управлять транзакциями (если только оператор вызова процедуры сам не находится внутри транзакции)

PL/pgSQL

- В качестве операторов в хранимых функциях и процедурах, написанных на языке PL/pgSQL, можно использовать операторы SQL
- Ни функции, ни процедуры не могут сохранять никакие значения во внутренних переменных между разными вызовами одной и той же или различных функций (процедур). Все данные, которые передаются между вызовами, должны быть переданы через параметры функций или процедур, конфигурационные параметры сервера базы данных или записаны в базу данных

PL/pgSQL и иные ЯП

- В системе PostgreSQL тело подпрограммы может быть записано на любом из языков программирования, известном серверу баз данных во время выполнения оператора, создающего функцию или процедуру
- По умолчанию это C и SQL – они считаются внутренними языками Postgres
- Также в базовой версии Postgres есть обработчики Python, Perl, Tcl
- Для иных языков существуют свои расширения

PL/pgSQL: функции: синтаксис

```
CREATE FUNCTION somefunc(integer, text) RETURNS integer
AS 'тело функции'
LANGUAGE plpgsql;
```

```
CREATE FUNCTION somefunc(integer, text) RETURNS integer
AS $$тело функции$$
LANGUAGE plpgsql;
```

PL/pgSQL: функции: синтаксис: блочная структура

Тело функции может быть также в виде блока:

```
AS $$
        [ DECLARE
                объявления ]
        BEGIN
                операторы
        END;
$$;
```

PL/pgSQL: функции: синтаксис: блочная структура

- DECLARE
  - необязательный раздел описаний локальных переменных, используемых в этом блоке
- BEGIN… END
  - раздел, содержащий выполнимые операторы

- Допускается также дополнительный раздел EXCEPTION, описывающий обработку «исключений»

PL/pgSQL: функции: синтаксис

Услов оператор записывается одним из следующих способов:

IF условие THEN
оператор; ...
ELSE
оператор; ...
END IF;
	IF условиеTHEN
	оператор; ...
	END IF;

PL/pgSQL: функции: синтаксис

Оператор выбора альтернативных вариантов вычислений может записываться в двух формах — с отдельными условиями на каждую альтернативу или с перечислением возможных значений выражения:

```
CASE
   WHEN условие-1 THEN
      оператор; ...
   WHEN условие-2 THEN
      оператор; ...
      ...
   ELSE
      оператор; ...
END CASE;
```

```
CASE выражение
   WHEN значение-1 THEN
      оператор; ...
   WHEN значение-2 THEN
      оператор; ...
      ...
   ELSE
      оператор; ...
END CASE;
```

PL/pgSQL: функции: синтаксис

- Создание цикла:

```
LOOP
        оператор; ...
END LOOP;
```

- Количество повторений цикла определяется с помощью следующих заголовков:
  - WHILE условие
  - FOR переменная IN начало .. конец BY шаг
  - FOREACH переменная IN ARRAY массив
- Цикл без заголовка повторяется бесконечно

PL/pgSQL: функции: пример

```
CREATE OR REPLACE FUNCTION hello(p text) RETURNS text
        LANGUAGE plpgsql AS $$
                DECLARE
                        v text;
                BEGIN
                        v := 'Hello, ‘;
                        RETURN v || p || '!’;
                END;
        $$;

SELECT hello('world');
```

PL/pgSQL: функции: пример

```
CREATE OR REPLACE FUNCTION example_fixed_loop()
RETURNS VOID AS $$
DECLARE
   i INT;
BEGIN
   RAISE NOTICE 'Цикл с фиксированным количеством повторений:';
   FOR i IN 1..5 LOOP
      RAISE NOTICE 'Итерация %', i;
   END LOOP;
END;
$$ LANGUAGE plpgsql;

SELECT example_fixed_loop();
```

PL/pgSQL: функции: пример

```
CREATE OR REPLACE FUNCTION example_array_loop()
RETURNS VOID AS $$
DECLARE
   j INT;
   numbers INT[] := ARRAY[1, 2, 3, 4, 5];
BEGIN
   RAISE NOTICE 'Цикл по элементам массива:';
   FOR j IN ARRAY_LOWER(numbers, 1)..ARRAY_UPPER(numbers, 1) LOOP
      RAISE NOTICE 'Элемент %: %', j, numbers[j];
   END LOOP;
END;
$$ LANGUAGE plpgsql;

SELECT example_array_loop();
```

PL/pgSQL: функции: пример

```
CREATE OR REPLACE FUNCTION example_if_else(num INT)
RETURNS TEXT AS $$
BEGIN
   IF num > 0 THEN
      RETURN 'Число положительное';
   ELSIF num < 0 THEN
      RETURN 'Число отрицательное';
   ELSE
      RETURN 'Число равно нулю';
   END IF;
END;
$$ LANGUAGE plpgsql;

SELECT example_if_else(5);  -- Возвращает 'Число положительное'
```

PL/pgSQL: функции: пример

```
CREATE OR REPLACE FUNCTION example_case(score INT)
RETURNS TEXT AS $$
BEGIN
   CASE
      WHEN score >= 90 THEN
         RETURN 'Отлично';
      WHEN score >= 80 THEN
         RETURN 'Хорошо';
      WHEN score >= 70 THEN
         RETURN 'Удовлетворительно';
      ELSE
         RETURN 'Неудовлетворительно';
   END CASE;
END;
$$ LANGUAGE plpgsql;

SELECT example_case(95);  -- Возвращает 'Отлично'
```

PL/pgSQL: процедуры: пример

```
CREATE OR REPLACE PROCEDURE create_user(new_email TEXT)
LANGUAGE plpgsql AS $$
BEGIN
   IF EXISTS (SELECT 1 FROM users WHERE email = new_email) THEN
      RAISE EXCEPTION 'Пользователь с таким email уже существует';
   END IF;

   INSERT INTO users(email)  VALUES (new_email)
   COMMIT;
   RAISE NOTICE ‘Адрес почты % добавлен', new_email;
EXCEPTION
   WHEN OTHERS THEN
      ROLLBACK;
END;
$$;
```

PL/pgSQL

- Но есть и потенциальные негативные последствия использования функций и процедур
  - Раздельное выполнение подзапросов. Вложенные запросы могут преобразовываться в операцию соединения. Однако подзапрос, размещенный внутри функции, оптимизируется и выполняется отдельно от основного запроса, в котором использована функция.
  - Побочные эффекты функций или процедур. База данных может измениться неочевидным и непредсказуемым для пользователя образом.
  - Недоступность оценок стоимости. Оптимизатор не всегда может рассчитать стоимость выполнения конкретной функции или процедуры, в связи с чем оптимизационные алгоритмы просто не сработают.

III. Тригеры

Триггеры

- Это хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: INSERT, UPDATE [OF имя_столбца [, ...]], DELETE, TRUNCATE
- По сути триггер – это функция, выполняющая действия в ответ на определенный набор событий

Триггеры

- Это хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: INSERT, UPDATE [OF имя_столбца [, ...]], DELETE, TRUNCATE
  - Существуют еще триггеры событий, которые не подключаются к конкретной таблице, а работают на уровне БД и позволяют создавать условия для выполнения команд DDL
- По сути триггер – это функция, выполняющая действия в ответ на определенное действие
- Триггеры могут существенно дополнить или изменить семантику стандартных операторов SQL. Например, триггеры можно использовать для проверки условий целостности, которые невозможно описать стандартными средствами языка SQL, или для регистрации изменений, выполняемых приложением, в другой таблице

Триггеры

- Для создания триггерной функции при определении хранимой процедуры нужно указать особый тип:

```
CREATE OR REPLACE FUNCTION log_user_updates()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$...$$;
```

Триггеры: синтаксис

CREATE [OR REPLACE] [CONSTRAINT] TRIGGER имя { BEFORE | AFTER | INSTEAD OF } { **событие** [OR ...] }
ON имя_таблицы
[FROM ссылающаяся_таблица]
[NOT DEFERRABLE | [DEFERRABLE] [INITIALLY IMMEDIATE | INITIALLY DEFERRED]]
[REFERENCING { { OLD | NEW } TABLE [AS] имя_переходного_отношения } [...]]
[FOR [EACH] { ROW | STATEMENT }]
[WHEN (условие)]
EXECUTE { FUNCTION | PROCEDURE } имя_функции (аргументы)

Допустимые **события**:

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

Триггеры: условия работы

- Триггерная функция должна вернуть либо NULL, либо запись/строку, соответствующую структуре таблице, для которой сработал триггер
- Предложения FOR EACH ROW, FOR EACH STATEMENT определяют, будет ли функция триггера срабатывать один раз для каждой строки, либо для SQL-оператора. Если не указано ничего, подразумевается FOR EACH STATEMENT (для оператора). Для триггеров ограничений можно указать только FOR EACH ROW

Триггеры: условия работы

Когда	Событие	На уровне строк	На уровне оператора
BEFORE	INSERT/UPDATE/DELETE	Таблицы и сторонние таблицы	Таблицы, представления и сторонние таблицы
	TRUNCATE	—	Таблицы и сторонние таблицы
AFTER	INSERT/UPDATE/DELETE	Таблицы и сторонние таблицы	Таблицы, представления и сторонние таблицы
	TRUNCATE	—	Таблицы и сторонние таблицы
INSTEAD OF	INSERT/UPDATE/DELETE	Представления	—
INSTEAD OF	TRUNCATE	—	—

Триггеры: специальные переменные

- Когда функция на PL/pgSQL срабатывает как триггер, в блоке (функции) верхнего уровня автоматически создаются несколько специальных переменных:
  - NEW - новая строка базы данных для команд INSERT/UPDATE в триггерах уровня строки. В триггерах уровня оператора и для команды DELETE эта переменная имеет значение NULL
  - OLD - старая строка 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 - имя сработавшего триггера
  - TG_WHEN - BEFORE, AFTER или INSTEAD OF в зависимости от определения триггера

Триггеры: специальные переменные

- Когда функция на PL/pgSQL срабатывает как триггер, в блоке (функции) верхнего уровня автоматически создаются несколько специальных переменных:
  - TG_LEVEL - ROW или STATEMENT в зависимости от определения триггера
  - TG_OP - операция, для которой сработал триггер: INSERT, UPDATE, DELETE или TRUNCATE
  - TG_TABLE_NAME - таблица, для которой сработал триггер
  - TG_TABLE_SCHEMA - схема таблицы, для которой сработал триггер
  - TG_NARGS - число аргументов в команде CREATE TRIGGER, которые передаются в триггерную функцию

Триггеры: примеры

```
CREATE TRIGGER check_update
    BEFORE UPDATE ON accounts
    FOR EACH ROW
    EXECUTE FUNCTION check_account_update();
```

Триггеры: примеры

```
CREATE OR REPLACE TRIGGER check_update
    BEFORE UPDATE OF balance ON accounts
    FOR EACH ROW
    EXECUTE FUNCTION check_account_update();
```

Триггеры: примеры

Триггер в следующем примере при любом добавлении или изменении строки в таблице сохраняет в этой строке информацию о текущем пользователе и отметку времени. Кроме того, он требует, чтобы было указано имя сотрудника и зарплата задавалась положительным числом.

Шаг 1: создаем таблицу:

```
CREATE TABLE emp (
    empname           text,
    salary            integer,
    last_date         timestamp,
    last_user         text
);
```

Триггеры: примеры

Шаг 2: создаем триггерную функцию:

```
CREATE FUNCTION emp_stamp() RETURNS trigger AS $emp_stamp$
    BEGIN
        -- Проверить, что указаны имя сотрудника и зарплата
        IF NEW.empname IS NULL THEN
            RAISE EXCEPTION 'empname cannot be null';
        END IF;
        IF NEW.salary IS NULL THEN
            RAISE EXCEPTION '% cannot have null salary', NEW.empname;
        END IF;

        -- Кто будет работать, если за это надо будет платить?
        IF NEW.salary < 0 THEN
            RAISE EXCEPTION '% cannot have a negative salary', NEW.empname;
        END IF;

        -- Запомнить, кто и когда изменил запись
        NEW.last_date := current_timestamp;
        NEW.last_user := current_user;
        RETURN NEW;
    END;
$emp_stamp$ LANGUAGE plpgsql;
```

Триггеры: примеры

Шаг 3: создаем триггер:

```
CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON emp
    FOR EACH ROW EXECUTE FUNCTION emp_stamp();
```