SQL на практике: создаем таблицу товаров и взаимодействуем с ней. Часть 3

Грачев Д.Г.

Процедуры

Процедура — объект базы данных, подобный функции, но имеющий следующие отличия:

- Процедуры определяются командой `CREATE PROCEDURE`, а не `CREATE FUNCTION`.
- Процедуры, в отличие от функций, не возвращают значение; поэтому в `CREATE PROCEDURE` отсутствует предложение `RETURNS`. Однако процедуры могут выдавать данные в вызывающий код через выходные параметры.
- Функции вызываются как часть запроса или команды DML, а процедуры вызываются отдельно командой `CALL`.
- Процедура, в отличие от функции, может фиксировать или откатывать транзакции во время её выполнения (а затем автоматически начинать новую транзакцию), если вызывающая команда `CALL` находится не в явном блоке транзакции.
- Некоторые атрибуты функций (например, `STRICT`) неприменимы к процедурам. Эти атрибуты влияют на вызов функций в запросах и не имеют отношения к процедурам.

Процедура vs Функция

Характеристика	ПРОЦЕДУРА	ФУНКЦИЯ
Возвращаемое значение	Нет (может быть OUT параметры)	Обязательно (одно или таблицу)
Вызов	CALL procedure()	SELECT function() или в SQL
Транзакции	Может содержать COMMIT/ROLLBACK	НЕ может управлять транзакциями
В запросах	Нельзя в SELECT, WHERE и т.д.	Можно использовать в SQL запросах
OUT параметры	Да	Да
IN параметры	Да	Да
INOUT параметры	Да	Да
Изменение данных	Любые операции	Обычно да, но с осторожностью

Синтаксис

```sql
-- функция

CREATE OR REPLACE FUNCTION function_name(param1 INT)

RETURNS INT AS $$

BEGIN

    RETURN param1 * 2;

END;

$$ LANGUAGE plpgsql;


-- Вызов:

  SELECT function_name(5);  -- 10
```

```sql
-- Процедура

CREATE OR REPLACE PROCEDURE procedure_name(param1
INT)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO logs VALUES (param1, now());
    COMMIT;
END;
$$;

-- Вызов:
  CALL procedure_name(5);
```

Таблицы

```sql
-- Категории продуктов

CREATE TABLE categories (

    id SERIAL PRIMARY KEY,

    name VARCHAR(100) NOT NULL,

    description TEXT,

    product_count INT DEFAULT 0,

    total_value DECIMAL(15,2) DEFAULT 0  --
суммарная стоимость всех продуктов

  );
```

```sql
-- Продукты
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(200) NOT NULL,
    category_id INT REFERENCES categories(id),
    price DECIMAL(10,2) NOT NULL,
    quantity INT DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
  )
```

```sql
CREATE OR REPLACE PROCEDURE transfer_product(
    INOUT p_product_id INT,           -- ID продукта (может измениться!)
    IN p_from_category INT,
    IN p_to_category INT,
    INOUT p_new_price DECIMAL DEFAULT NULL  -- новая цена (опционально)
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_old_category INT;
    v_old_price DECIMAL;
BEGIN
    -- Получаем текущие данные
    SELECT category_id, price INTO v_old_category, v_old_price
    FROM products WHERE id = p_product_id;
    IF NOT FOUND THEN
        RAISE EXCEPTION 'Продукт % не найден', p_product_id;
    END IF;
    -- Проверяем категории
    IF v_old_category != p_from_category THEN
        RAISE EXCEPTION 'Продукт не в категории %', p_from_category;
    END IF;
    -- Обновляем продукт
    UPDATE products
    SET category_id = p_to_category,
        price = COALESCE(p_new_price, v_old_price)
    WHERE id = p_product_id;
    -- Обновляем статистику категорий
    UPDATE categories SET product_count = product_count - 1,
        total_value = total_value - v_old_price * quantity
    WHERE id = p_from_category;
    UPDATE categories SET product_count = product_count + 1,
        total_value = total_value + (COALESCE(p_new_price, v_old_price) * quantity)
    WHERE id = p_to_category;
    COMMIT;
    RAISE NOTICE 'Продукт % перемещен из % в %',
        p_product_id, p_from_category, p_to_category;
END;
$$;
```

Более продвинутый пример

```
CREATE OR REPLACE PROCEDURE recalculate_all_categories()
LANGUAGE plpgsql
AS $$
DECLARE
    cur_cursor CURSOR FOR SELECT id FROM categories;
    v_category_id INT;
BEGIN
    OPEN cur_cursor;
    LOOP
        FETCH cur_cursor INTO v_category_id;
        EXIT WHEN NOT FOUND;

        -- Обновляем статистику каждой категории
        UPDATE categories SET
            product_count = (SELECT COUNT(*) FROM products WHERE category_id = v_category_id),
            total_value = (SELECT COALESCE(SUM(price * quantity), 0)
                           FROM products WHERE category_id = v_category_id)
        WHERE id = v_category_id;

        COMMIT;  -- Коммитим каждую категорию отдельно!
    END LOOP;
    CLOSE cur_cursor;
END;
    $$;
```

Аспект	ПРОЦЕДУРА	ФУНКЦИЯ
Появилась в	PostgreSQL 11	Всегда
RETURN	Необязательно (OUT)	Обязательно
Вызов	`CALL`	`SELECT`
Транзакции	✅ Есть контроль	❌ Нет контроля
В запросах	❌ Нельзя	✅ Можно везде
EXCEPTION	✅ Полный контроль	✅ Только в BEGIN...END
Производительность	Выше для действий	Выше для вычислений
Множество результатов	Через REFCURSOR	Через RETURNS TABLE
Изменение данных	Любые	Ограниченно

Вывод

ПРОЦЕДУРЫ	ФУНКЦИИ
📦 Действия	🧮 Вычисления
💾 Изменение данных	📊 Возврат данных
🔄 Управление транзакциями	🔍 Часть SQL запросов
🏭 Бизнес-процессы	📈 Отчеты и расчеты
⚙️ Массовые операции	🔧 Утилиты и хелперы

- Функции - для вычислений и данных
- Процедуры - для действий и процессов
- Триггеры - для автоматической реакции на события

Практическое задание

```
-- Процедура: списание товаров (с транзакцией!)

CREATE PROCEDURE write_off_products(

    p_product_id INT,

    p_quantity INT,

    p_reason VARCHAR

);


-- Функция: проверка остатков

CREATE FUNCTION check_stock_level(

    p_category_id INT,

    p_threshold INT DEFAULT 10

   ) RETURNS TABLE(product_id INT, name VARCHAR, current_quantity INT);
```

Создайте процедуру и функцию для инвентаризации.
При процедуре списания входязие параметры: продукт, кол-во, причина (создать таблицу логов)
Для функции описать для проверки что в данной категории именно столько продуктов, сколько переданно в функцию.

Ресурсы

- https://postgrespro.ru/docs/postgresql/current/xproc
- https://postgrespro.ru/docs/postgresql/current/xfunc-sql