

Тема 1. Знакомство с серверным программированием

Основные понятия

Серверное программирование — программирование бизнес-логики средствами сервера баз данных.

Пользовательская функция (User Function) — это именованный блок кода, который написан пользователем базы, выполняет заданные действия и возвращает значение.

Пользовательская процедура (User-Defined Procedures) — это тоже именованный блок кода, который хранится в базе данных и может быть вызван по запросу для выполнения определённой операции или набора операций.

Подпрограммы (ROUTINE) — общее именование функций и процедур.

Процедурный язык (англ. PL, Procedural Language) — язык, позволяющий писать подпрограммы в императивном стиле внутри сервера баз данных.

Триггер — это специальный объект, который автоматически срабатывает, когда с одной из таблиц БД происходит определённое событие.

Отличия функций и процедур

Критерий	Функции	Хранимые процедуры
Возвращаемое значение	Обязательно возвращает значение — одиночное значение заданного типа (число, строка или таблица).	Как правило, не возвращает значение. Может выдавать результат через выходные параметры.
Параметры	В большинстве случаев имеет входные параметры. Выходных обычно не имеет.	Может иметь и входные, и выходные параметры.
Транзакции	Нет возможности управлять транзакциями внутри кода функции.	Можно управлять транзакциями внутри процедуры при помощи команд <code>commit</code> и <code>rollback</code> .
Вызов	Вызываются как часть запроса, команды DML или DQL. Например, функцию можно использовать внутри запросов <code>SELECT</code> и <code>UPDATE</code> .	Процедуры вызываются отдельно командой <code>CALL</code> . Нельзя вызвать процедуру из <code>SELECT</code> или <code>UPDATE</code> запросов.
Использование результата	Результат функции можно использовать для дальнейших преобразований, фильтрации и возврата данных в результате запроса.	Если процедура возвращает значение, его можно только присвоить переменной на стороне вызывающего кода.

Тема 2. Создание процедур и функций

Пользовательская функция

```
CREATE [OR REPLACE] FUNCTION имя_функции(список_аргументов)
RETURNS тип_возвращаемого значения
LANGUAGE язык
AS $$ тело_функции $$
```

Вызвать её можно с помощью запроса **SELECT**.

Пользовательская процедура

```
CREATE [OR REPLACE] PROCEDURE имя_процедуры(список_аргументов)
LANGUAGE язык
AS $$ тело_процедуры $$
```

Процедура с тремя входными параметрами:

```
CREATE OR REPLACE PROCEDURE add_workout_data(in integer, in integer, in numeric)
LANGUAGE sql -- Обозначаем язык
AS $$

    -- Вставляем данные в таблицу physiological_indicators
    INSERT INTO physiological_indicators(workout_id, date_time, heart_rate)
    VALUES ($1, transaction_timestamp(), $2);

    -- Вставляем данные в таблицу distances
    INSERT INTO distances(workout_id, date_time, distance)
    VALUES ($1, transaction_timestamp(), $3);

$$
```

Объявление переменных

```
CREATE OR REPLACE FUNCTION/PROCEDURE имя(список_аргументов_и_их_типы)
RETURNS тип_возвращаемого_значения -- только для функции
LANGUAGE plpgsql
AS $$

DECLARE
    имя_переменной_1 тип_переменной_1;
    имя_переменной_2 тип_переменной_2;
BEGIN
    тело_процедуры_или_функции;
END;
$$
```

Тема 2. Создание процедур и функций

Обработка исключений EXCEPTION

```
BEGIN
    -- Основной код
EXCEPTION
    WHEN -- Условие
        THEN -- Код обработки исключения
    WHEN -- Условие
        THEN -- Код обработки исключения
END;
```

GET STACKED DIAGNOSTICS — запрос дополнительной информации об ошибке.

Анонимный блок

```
DO
LANGUAGE plpgsql
$$
DECLARE
...
-BEGIN
...
END;
$$
```

С помощью анонимных блоков записывают промежуточные результаты вычисления значений в отдельные переменные. Алгоритм такой:

1. Объявить переменные с помощью **DECLARE** — как в примере из урока:

```
DO
LANGUAGE plpgsql
$$
DECLARE
    _avg_distance_with_group numeric; -- Средняя дистанция пользователей в группах
    _avg_distance_without_group numeric; -- Средняя дистанция
                                         -- самостоятельных спортсменов
BEGIN
...
END;
$$
```

2. Определить значение каждой из этих переменных с помощью SQL-запросов.
3. Вывести результат сравнения с помощью **RAISE NOTICE**.

Тема 3. Параметры подпрограмм

Типы параметров

- **IN** (входные) — в них передаются значения при вызове подпрограммы. Практически во всех примерах, которые вы рассматривали раньше, были только входные параметры. Их обозначают ключевым словом IN, но можно не обозначать никак — это тип параметров по умолчанию.
- **OUT** (выходные) — их значения вычисляются внутри подпрограммы и при её вызове передаются наружу. Они позволяют расширить диапазон значений, которые возвращают подпрограммы. Их обозначают ключевым словом OUT перед названием параметра или его типом.
- **INOUT** (одновременно входные и выходные) — в них передаётся значение, которое обрабатывается в теле процедуры, а затем результат этой обработки передаётся наружу.
- **VARIADIC** (переменные) — необязательные параметры, при вызове подпрограммы могут не передаваться в неё. В этой теме переменные параметры вы рассматривать не будете, подробнее изучить их можно [в официальной документации](#).

Именованные параметры:

```
CREATE OR REPLACE FUNCTION check_speed(p_speed numeric, p_user_id integer)
```

Присвоение параметру значения по умолчанию:

```
(param_1 тип_параметра DEFAULT значение_по_умолчанию)
```

Тема 4. Управляющие конструкции

Оператор ветвления IF

```
IF логическое_выражение THEN  
    операторы_1;  
ELSE  
    операторы_2;  
END IF;
```

Здесь операторы 1 будут выполнены, если логическое выражение истинно, а операторы 2 — если оно ложно.

Оператор выбора CASE

Синтаксис этого оператора схож с синтаксисом оператора IF.

```
CASE  
    WHEN < условие > THEN  
        < основной блок команд >  
    ELSE  
        < альтернативный блок команд >  
END CASE;
```

Тема 4. Управляющие конструкции

Циклы и команды

Команды	Назначение	Запрос
Цикл <code>FOR</code>	<ul style="list-style-type: none"> - для итерации результатов запроса - для итерации целых чисел в заданном интервале 	<code>FOR _г IN запрос</code> <code>FOR _i IN интервал</code>
Цикл <code>FOREACH</code>	для итерации по элементам массива	<code>FOREACH _el IN ARRAY</code>
Цикл <code>WHILE</code>	для выполнения тела цикла по условию, например, суммирование случайных чисел, пока не достигнута нужная сумма	<code>WHILE условие</code>
Команда <code>EXIT</code>	для остановки цикла	<code>EXIT WHEN условие_выхода;</code> — выход из цикла по условию <code>EXIT;</code> — безусловное прекращение цикла
Команда <code>CONTINUE</code>	для остановки выполнения тела цикла и вызова новой итерации	<code>CONTINUE WHEN условие;</code> — переход к следующей итерации при достижении условия <code>CONTINUE;</code> — безусловный переход к следующей итерации

Тема 5. Триггеры

На какие события может реагировать триггер:

- **INSERT** — вставка новых строк в таблицу;
- **UPDATE** — изменение существующих строк;
- **DELETE** — удаление строк из таблиц;
- **TRUNCATE** — очистка таблицы.

Время срабатывания задают одним из трёх ключевых слов:

- **BEFORE** — до выполнения операции. Например, до вставки новой записи.
- **AFTER** — после выполнения операции. Например, после обновления записи.
- **INSTEAD OF** — вместо выполнения операции. Например, вместо удаления записи.

Способ срабатывания триггера задаёт, сколько раз вызовется триггерная функция, если запрос задействует несколько строк:

- **FOR EACH ROW** (для каждой строки) — функция вызовется столько раз, сколько строк таблицы участвует в запросе. Например, для каждой вставленной строки.
- **FOR EACH STATEMENT** (для каждого выражения) — функция вызовется один раз для каждого запроса. Например, один раз для всей вставки.

Создание триггера:

```
CREATE OR REPLACE TRIGGER имя_триггера
BEFORE/AFTER/INSTEAD OF -- время срабатывания относительно события
INSERT/UPDATE/DELETE -- триггерное событие
ON имя_таблицы -- таблица, на события в которой реагирует триггер
FOR EACH ROW / FOR EACH STATEMENT -- вариант реагирования
EXECUTE FUNCTION trigger_function_name(); -- функция, которую вызывает триггер
```

Создание триггерной функции:

```
CREATE OR REPLACE FUNCTION trigger_function()
RETURNS trigger
LANGUAGE plpgsql
AS $$
BEGIN
/*
    тело функции
*/
    RETURN record/NULL; -- возвращаемое значение record или NULL
END;
$$;
```

Тема 5. Триггеры

Доступ к данным изменяемой, добавляемой или удаляемой строки, триггерная функция получает с помощью двух переменных с типом данных **record**: **NEW** и **OLD**.

Значения переменных **NEW** и **OLD** зависят от типа события:

	OLD	NEW
INSERT	[null]	вставляемая строка
UPDATE	изменяемая строка, до изменений	изменённая строка, после изменений
DELETE	удаляемая строка	[null]

Специальные триггерные переменные:

Переменная	Значение
TG_NAME	имя сработавшего триггера
TG_WHEN	строка, в зависимости от времени срабатывания триггера содержит BEFORE , AFTER или INSTEAD OF
TG_LEVEL	строка, в зависимости от типа триггера содержит ROW или STATEMENT
TG_OP	строка, в зависимости от операции, для которой сработал триггер, содержит INSERT , UPDATE , DELETE или TRUNCATE
TG_TABLE_NAME	имя таблицы, для которой сработал триггер
TG_TABLE_SCHEMA	имя схемы, где находится таблица, для которой сработал триггер