Procedurální programování

Procedurální programování je podobné funkcionálnímu. Výpočet probíhá sekvencí procedur, což jsou jednoduché funkce, které jsou v programu globálně definovány a jsou vždy k dispozici. Výpočet se potom sestává z postupném volání těchto procedur a předáváním výsledku. Často uváděné vlastnosti procedurálního programování jsou: lineární výpočet, vedlejší efekty, jasně stanovený postup kroků. Procedury jsou jakési základní funkce, které mají jednodušší vnitřní strukturu. Cílem je použit již definované procedury při co nejvíce příležitostech. Jsou obecně použitelné kdekoli v programu. Funkce pracují s hodnotami, a vytváří na základě vstupu nové a sami funkce se používají jako hodnoty. Často se buduje abstrakce a vnitřně se využívá dalších funkcí k dosažení výsledku. První implementace můžeme najít v jazycích jako je ALGOL, Fortran, COBOL aj. Často se využívá principu modularity, rozdělení a oddělení jednotlivých větších celků programu do modulů.

PL/pgSQL

Je Procedural Language/PostgreSQL, procedurální nadstavba Postgre. V mnohém je podobná Oraklovské PL/SQL. Zajímavé rozdíly oproti dalším procedurálním jazykům:

- pg nemá defaultní hodnoty
- v pg lze přetěžovat názvy procedur
- namísto packagů používá schémata

Společně potom ovlivnily SQL/PSM, která přináší určité další rozdíly (práci s výjimkami) a je definována ISO standartem. Všechny tři vychází z Ada PL, které je založená na Pascalu.

PL/pgSQL byl vytvořen v roce 1988 Jan Weickem pod hlavičkou PostgreSQL Global Development Group. Jeho první verze se objevila v Postgre 6.4. Stejně jako samotné Postgre jde o opensource projekt. Jedná se o jednoduchý programovací jazyk navržený pouze pro psaní procedur.

Výhody a nevýhody PL/pgSQL

Při běžném používání pgSQL musí být každý dotaz odeslán na server, vyhodnocen a poté musí server odeslat zpátky odpověď a může následovat další dotaz. PL umožňuje shlukovat jednotlivé bloky dotazů na serveru, a provádět výpočty bez další komunikace s klientem. Eliminuje se tak zbytečná komunikace. Mezivýsledky, které klient nepotřebuje se vůbec neodesílají ale rovnou se použijí pro další výpočet. PL/pgSQL umožňuje definovat procedury, které se dají na mnoha místech volat.

Syntaxe jazyka

V jazyce můžeme definovat funkce, nebo procedury. V našem základním pohledu je rozdíl v tom, že funkce vrací hodnotu (pomocí klíčového slova RETURN), zatímco procedura hodnotu nevrací. Ostatně jak už bylo zmíněno, tak právě procedurální programování povolují tzv. vedlejší efekt. V příkladech se budeme zabývat vytvářením funkcí. Pro práci s procedurami jen nahradím klíčové slovo FUNCTION slovem PROCEDURE a samozřejmě nebudeme vracet výslednou hodnotu.

Při psaní definujeme funkce, které obsahují hlavičku, štítek a dvě hlavní části, a to *deklarační část* a *tělo funkce*. Základní představa o funkci by tedy mohla vypadat následovně:

hlavička
deklarační část;
BEGIN
tělo funkce;
END;
štítek
	hlavička
	BEGIN
	tělo funkce
	END;
	štítek

/*ukázka 1*/	***/*ukázka 2*/***

Deklarační část přitom můžeme u jednodušších funkcí vynechat viz. druhá ukázka. Jednotlivé parametry funkce jsou přístupné podle pořadí v hlavičce jako $1, $2,… Hlavička funkce začíná buď klíčovým slovem CREATE pro tvorbu nové funkce nebo ‚REPLACE' pro změnu již existující funkce. Pokud nevíme, zda je již funkce vytvořená dá se použít kombinace CREATE OR REPLACE. Její vyhodnocení je poté očividné. V hlavičce také uvádíme název funkce, parametry a jejich typy a návratovou hodnotu funkce a její typ. Pro naši základní představu budeme používat slovo $$, i když je možno také funkci samotnou ohraničit pouze uvozovkami (to jen pro zajímavost). Ve štítku poté uvádíme jazyk, ve kterém je funkce deklarována. Zde tedy ukázka, jak by mohla vypadat obecně zapsaná deklarace nové funkce:

CREATE FUNCTION *název_funkce***(***název_parametru typ_parametru***, ….) RETURNS** typ **AS $$**
    DECLARE *deklarace_proměnných*
    BEGIN
        blok příkazů/dotazů;
        RETURN *výraz/proměnná***;**
    END;
$$ LANGUAGE plpgsql;

/*ukázka 3*/

V deklarační části můžeme deklarovat nové proměnné, aliasy parametrů, nebo relační proměnné. U deklarace relační proměnné můžeme požít klíčové slovo %TYPE pro získání stejného typu. Pro lepší představu konkrétní ukázka, jak mohou vypadat různé deklarace.

Deklarace nové proměnné:	cislo integer DEFAULT 0;
Deklarace aliasu parametru:	id_zamestnance ALIAS FOR $1;
Deklarace relační proměnné:	jmeno studentiUPOL.jmeno%TYPE;

Řízení výpočtu

Už umíme vytvořit základní funkci, která nám vyhodnotí postupně krok po kroku své tělo a vrátí nám hodnotu. Při reálném použití se nám ale velmi hodí upravit toto vyhodnocování přeskočením určitých částí, nebo jejich opakování. K tomu nám slouží podmínky a cykly.

Volání již vytvořené funkce

Pro volání již vytvořené funkce použijeme klíčové slovo PERFORM. Z výsledek můžeme pracovat jako s již vypočítanou hodnotou.

Podmínky

K vytvoření podmíněných částí programu používáme již dobře známe slovo IF, které ale následuje slovo THEN, jako například v jazyku PASCAL, poté můžeme požít ELSE, nebo opět větvit výpočet pomocí ELSEIF. Pro konec podmínky použijeme END IF, kterým celé větvení zakončíme.

IF *podmínka* **THEN**
výrazy
ELSE
výrazy
END IF;
/*ukázka 4*/

Pokud bychom chtěli použít vícekrát ELSEIF měli bychom se zamyslet, zda by nebylo přehlednější využít možnosti CASE. Existují dva způsoby jeho použití v ukázce 5 vidíte tzv. jednoduchý case. Ten vyhodnotí hledaný výraz a poté porovnává jednotlivé případy pod slovy WHEN, pokud se případ rovná výsledku hledanému výrazu je provedeno jeho tělo. Druhou možností je case bez hledaného výrazu. Program poté prochází jednotlivé případy, ve kterých musí být výraz typu boolean, pokud je výraz True tělo se vyhodnotí, pokud False, tak se přeskočí. Celý case ukončíme použitím klíčového slova END CASE.

CASE *hledaný-výraz*
WHEN *výraz* **THEN**
příkazy
WHEN *výraz* **THEN**
příkazy
.
.
.
ELSE
příkazy
END CASE;
/*ukázka 5*/

CASE
WHEN boolean-výraz THEN
příkazy
WHEN boolean-výraz THEN
příkazy
.
.
.
ELSE
příkazy
END CASE;
/*ukázka 6*/

Cykly

K opakování určité části programu nám slouží různé typy cyklů. LOOP slouží k nepodmíněnému cyklení, které skončí pouze pokud provede RETURN, EXIT, nebo CONTINUE. Klíčové slovo EXIT zkontroluje výraz typu boolean a pokud je pravdivý ukončí cyklus a program pokračuje hned za ním. Rozdílným případem je příkaz CONTINUE, ten také zkontroluje výraz typu boolean, ale pokud je pravdivý neukončí celý cyklus, ale provede další iteraci cyklu (přeskočí tedy část cyklu za tímto příkazem). Nejjednodušším podmíněným cyklem je cyklus WHILE, který vždy před začátkem iterace, zkontroluje výraz typu boolean. Pokud je výraz True provede se daná iterace, v opačném případě se cyklus ukončí. Jen podotknu, že i v podmíněných cyklech můžeme použít příkazy EXIT a CONTINUE. Všechny typy cyklů opět musíme ukončit klíčovým slovem END LOOP.

LOOP

příkazy

EXIT WHEN *podmínka*;

CONTINUE WHEN *podmínka*;

příkazy

END LOOP;

/*ukázka 7*/

WHILE *boolean-výraz* **THEN**

příkazy

END LOOP;

WHILE NOT *boolean-výraz* **THEN**

příkazy

END LOOP;

/*ukázka 8 – dva cykly while*/

Asi nejpoužívanějším typem cyklů je podmíněný cyklus FOR, ten při každé iteraci zvedá hodnotu předem definovanou v jeho hlavičce. Pokud navíc použijeme slovo REVERSE, hodnotu nezvedá, ale snižuje. Pokud bychom chtěli zvedat, nebo snižovat hodnotu o více než o jedna, použijeme klíčové slovo BY. Například 1..10 BY 2 by provedlo iterace pro 1, 3, 5, 7 a 9. Dále jen zmíním, že pro pohyb v datových strukturách máme také cyklus FOREACH, tím se ale v naší ukázce zabývat nebudeme. Pro zájemce je na konci dokumentu odkaz pro více informací.

FOR *proměnná* **IN** *rozsah* **LOOP**

příkazy

END LOOP;

/*ukázka 9*/

FOR *proměnná* **IN REVERSE** *rozsah* **LOOP**

příkazy

END LOOP;

/*ukázka 10*/

Konkrétní příklady – základní aplikace poznatků

1. Funkce vrací součet dvou čísel typu integer.

```
CREATE FUNCTION soucet_dvou(a integer, b integer) RETURNS integer AS $$
    BEGIN
        RETURN a + b;
    END;
$$ LANGUAGE plpgsql;
```

2. Funkce vrací příjmení zaměstnanců z tabulky, u kterých se křestní jméno shoduje s námi zadaným jménem

```
CREATE FUNCTION zamestnanci(jmeno text) RETURNS text AS $$
    DECLARE hledane_jmeno ALIAS FOR $1;
        vsechna_prijmeni zamestnanci.prijmeni;
    BEGIN
        RETURN SELECT INTO vsechna_prijmeni prijmeni FROM zamestnanci
                                    WHERE krestni = hledane_jmeno;
    END;
$$ LANGUAGE plpgsql;
```

3. Fuknce signum napsaná v PL/pgSQL

```
CREATE FUNCTION sgn(cislo integer) REUTRNS integer AS $$
        BEGIN
                CASE
                        WHEN cislo > 0 THEN RETURN 1;
                        WHEN cislo < 0 THEN RETURN -1;
                        ELSE RETURN 0;
        END;
$$ LANGUAGE plpgsql;
```

4. Funkce n-té Fibonacciho číslo pomocí cyklu

```
CREATE FUNCTION fib1(cislo integer) RETURNS integer AS $$
        DECLARE predminule integer := 0;
                  minule integer := 1;
                  soucasna integer := 0;
        BEGIN
                IF cislo < 2 THEN
                        RETURN cislo;
                END IF;
                FOR i IN 2..cislo LOOP
                        soucasna := predminula + minula;
                        predminula := minula;
                        minula := soucasna;
                END LOOP;
                RETURN soucasna;
        END;
$$ LANGUAGE plpgsql;
```

5. Funkce n-té Fibonacciho číslo rekurzivně

```
CREATE FUNCTION fib2(cislo integer) RETURNS integer AS $$
        BEGIN
                IF cislo < 2 THEN
                        RETURN cislo;
                ELSE
                        RETURN PERFORM fib2(cislo - 1) + PERFORM fib2(cislo - 2);
                END IF;
        END;
$$ LANGUAGE plpgsql;
```

Vypsání hodnoty

Především při tvorbě procedur by se nám hodilo vypsat hodnotu na obrazovku. K tomu slouží:

RAISE NOTICE ‘*hodnota: %*‘**,** proměnná;

Dokumentace

PostgreSQL: Documentation: 12: Chapter 42. PL/pgSQL - SQL Procedural Language. PostgreSQL: The world's most advanced open source database [online]. Copyright © 1996 [cit. 16.03.2022]. Dostupné z: https://www.postgresql.org/docs/12/plpgsql.html