Eigene Funktionen innerhalb von pgSQL

Michael Dienert

25. November 2015

1 Warum noch eine Programmiersprache?

SQL ist ein Standard, der von praktisch allen relationalen Datenbank-Servern als Abfragesprache verwendet wird [1] . Der Nachteil von SQL ist, dass jedes *SQL-Statement* einzeln vom Datenbankserver ausgeführt werden muss.

Das bedeutet, eine Datenbankanwendung muss diese jede Abfrage zum Server schicken, auf das Ergebnis warten, evtl. die empfangenen Daten bearbeiten, die nächste Abfrage senden, usw.

Diese strenge Trennung in Client- und Serveraufgaben - SQL auf dem Server, Aufbereitung der SQL-Ergebnisse auf dem Client - ist langsam und erzeugt, falls Server- und Client auf verschiedenen Rechner laufen, auch unnötigerweise viel Netzlast.

Dieses Problem kann man vermeiden, wenn man die Bearbeitung der Abfrageergebnisse bereits auf dem Server durchführt. Da sich reines SQL nur schlecht dazu eignet, allgemeine Algorithmen zu programmieren, kann man Datenbankserver wie z.B. postgres und Oracle um Programmiersprachen erweitern.

Eine dieser Programmiersprachen ist PL/pgSQL, die der bei Oracle verwendeten Sprache PL/SQL sehr ähnlich ist.

2 Hinzufügen von PL/pgSQL zu einer Datenbank

Bei einer neu erzeugten Datenbank ist PL/pgSQL noch nicht verfügbar. Es muss zunächst in die Datenbank geladen werden.

```
CREATE LANGUAGE 'plpgsql';
```

Dieses Anweisung erzeugt einen Fehler, wenn pspgsql schon geladen ist. Mit einer etwas umständlichen sql-Abfrage lässt sich das vermeiden. Die Syntax dazu ist im Anhang.

3 Ein Beispiel mit PL/pgSQL

Um das Laden von PL/pgSQL zu erleichtern und eine Testdatenbank bereitzustellen, könnt Ihr mit

[1] Ein Server ist in diesem Zusammenhang keine Maschine, sondern ein Programm, das im Hintergrund läuft und an einem bestimmten Port (TCP-Port 5432 bei postgres) auf SQL-Abfragen wartet.

```
wget http://dt.wara.de/itKlassen/e2it2/functions.sql
```

ein SQL-Skript vom msv-Webserver beziehen.

Hier nocheinmal die üblichen Schritte die nötig sind, um dieses Skript einzuspielen. Achtung! Mein Textsatzprogramm macht aus einfachen Hochkommas sog. *Backticks*. An deren Stelle muss man aber Hochkomma versenden:

```
su postgres ... Passwort ist 'postgres'
createuser <meinAccountname>
exit
psql -f functions.sql postgres
```

4 Eine sehr kurze Einführung in PL/pgSQL

4.1 CREATE FUNCTION

Eine PL/pgSQL -Funktion hat grundsätzlich diesen Aufbau:

```
CREATE FUNCTION <funktionsname> (<funktions-parameter>)
  RETURNS <rueckgabetyp> AS
'
  DECLARE
    deklarationen;
    ...
  BEGIN
    anweisungen;
    ...
  END;
'
LANGUAGE 'plpgsql';
```

Das kann man auch ein bisschen zusammenfassen und dann sieht es so aus:

```
CREATE FUNCTION <fn-name> (<args>) RETURNS <dat-typ> AS ' ... ' LANGUAGE 'plpgsql';
```

D.h. die eigentliche Funktionsdefinition ist lediglich ein Textstring. Damit hat man nun folgendes Problem: jedes Hochkomma innerhalb des Textstrings würde den begonnenen String abschliessen. Damit das nicht passiert, muss man Hochkommas innerhalb der Funktion *verdoppeln*. Da das aber zu absolut unleserlichem Code führt (es gibt Beispiele mit bis zu 10 Hochkommas hintereinander), sollte man den Funktions-String mit der Postgres *Dollar-Quotation* begrenzen:

```
CREATE FUNCTION <fn-name> (<args>) RETURNS <dat-typ> AS $$ ... $$ LANGUAGE 'plpgsql';
```

Zwischen die beiden Dollar-Zeichen kann man auch noch eine beliebige Markierung (Tag) einführen:

```
$tt$ ... $tt$
```

4.2 Aufgabe

Füge der Datei `functions.sql` dieses CREATE FUNCTION Codegerüst hinzu.

Überlege Dir dazu einen Namen für die Funktion.

Die Funktion, die wir schreiben wollen, soll alle CDs **einer** Band in einen Textstring kopieren. Die CD-Titel stehen in der Tabelle 'cd'.

Frage: welchen Parameter muss ich meiner Funktion übergeben, damit diese in der Tabelle 'cd' die cd-Titel einer Band herausfinden kann?

4.3 Der Deklarationsteil

PL/pgSQL sieht der Programmiersprache Pascal ähnlich. Deklarationen werden so geschrieben:

```
book_price FLOAT;
book_title TEXT;
```

4.3.1 Der Datentyp `%ROWTYPE`

`%ROWTYPE` ist ein spezieller Datentyp, der ganze *Datensätze* aufnehmen kann. Man benötigt ihn, wenn man das Ergebnis einer SELECT-Abfrage speichern möchte.

Hier die Deklaration einer `%ROWTYPE`-Variablen für die Zeilen der Tabelle **`cd`**:

```
DECLARE
   zeile cd%ROWTYPE;
```

Mit `SELECT INTO` kann das Ergebnis einer SELECT-Abfrage der `%ROWTYPE`-Variablen zugewiesen werden.

4.4 Aufgabe

Füge der Funktion eine Deklaration für eine Textvariable und eine `%ROWTYPE`-Variable für Zeilen der Tabelle 'cd' hinzu.

4.5 Ausdrücke und Kommentarzeilen

Auch die Zuweisung in PL/pgSQL entspricht der Pascal-Syntax. Einzeilige Kommentare beginnen jedoch wie in SQL üblich mit '–':

```
-- preiserhoehung um 20 prozent:
book_price := book_price * 1.2;
```

4.6 IF/THEN/ELSE-Bedingung

> Auch hier wird wieder eine PASCAL-ähnliche Syntax verwendet! Das heisst, im Unterschied zu C und Java steht die Bedingung (condition) *nicht* in runden Klammern und der Test auf Gleichheit ist ein einfaches '=' - Zeichen !!!!

Eine Fallunterscheidung mit IF/THEN/ELSE sieht so aus (Der ELSE-Teil ist optional):

```
CREATE FUNCTION identifier (arguments) RETURNS type AS '
  DECLARE
    declarations
```

```
  BEGIN

    IF id < 100 THEN

      ...

    ELSE

      ...

    END IF;

  END;
' LANGUAGE 'plpgsql';
```

Für eine Serie von Fallunterscheidungen gibt es auch noch `ELSE IF`.

4.7 Schleifen

4.7.1 Die `WHILE`-Schleife

Syntax der `WHILE`-Schleife:

```
WHILE i < 10 LOOP

   ...

END LOOP;
```

4.7.2 Die FOR-Schleife

Beispiel einer einfachen Zählschleife:

```
FOR i IN 0..15 LOOP

   ...

END LOOP;
```

Wesentlich mächtiger ist folgende FOR-Schleife, die über alle Zeilen einer Abfrage läuft, so eine Art **forEach**-Schleife:

```
FOR zeile IN SELECT * FROM cd LOOP

   ...

END LOOP;
```

`zeile` muss dabei vom Typ `%ROWTYPE` sein. Siehe oben.

4.8 Aufgabe

Mit welcher SQL-Abfrage würde man aus der Tabelle `cd` alle Zeilen einer bestimmten Band selektieren?

Welche Anweisung muss im Rumpf der Schleife stehen, damit in einer Textvariablen die CD-Titel mit Komma getrennt aneinander gefügt werden?

Hinweis: der Verkettungsoperator in PL/SQL sieht so aus: `||` . Er funktioniert so wie das '+'-Zeichen bei Java-Strings.

4.9 Aufruf der Funktion

Unsere nun selbstgeschriebene Funktion kann jetzt in einer SELECT-Abfrage verwendet werden. Teste sie mit folgender Abfrage (der Name der Funktion ist natürlich anzupassen):

```
SELECT band_name, cdListe(band_id) FROM band;
```

In der CD-Liste stört noch ein überflüssiges Komma am Anfang oder Ende der Liste. Vermeide dieses Komma durch Einfügen einer Fallunterscheidung in die FOR-LOOP.

```
DROP DATABASE IF EXISTS fun;
CREATE DATABASE fun;
\connect fun

-- prozedurale sprache laden
-- ################################################################
-- fuerchterlicher hack um die fehlermeldung zu vermeiden die
-- entsteht, wenn plpgsql geladen werden soll, obwohl es schon
-- geladen ist

-- die folgende funktion wird nur erzeugt, um plpgsql aus einem sql-
-- statement heraus laden zu koennen:
CREATE OR REPLACE FUNCTION create_language_plpgsql()
RETURNS BOOLEAN AS $$
    CREATE LANGUAGE plpgsql;
    SELECT TRUE;
$$ LANGUAGE SQL;

-- hier wir geprueft, ob plpgsql schon da ist. wenn nicht, wird es
-- geladen
SELECT CASE WHEN NOT
    (
        SELECT  TRUE AS exists
        FROM    pg_language
        WHERE   lanname = 'plpgsql'
        UNION
        SELECT  FALSE AS exists
        ORDER BY exists DESC
        LIMIT 1
    )
THEN
    create_language_plpgsql()
ELSE
    FALSE
END AS plpgsql_created;

-- die funktion wird im folgenden nicht mehr benoetigt und kann weg
DROP FUNCTION create_language_plpgsql();

-- ################################################################
```

```
-- ein ganz einfaches er-modell

--  -------- 1        ------
--  | band | -------- | cd |
--  --------        n ------


-- eine tabelle mit ein paar musikkapellen
CREATE TABLE band (band_id INTEGER, band_name TEXT);
copy band from stdin with delimiter as '&';
10&The White Stripes
20&The Who
30&The Rolling Stones
40&Spliff
50&Slickaphonics
\.

--eine tabelle mit ein paar musik-cds
CREATE TABLE cd (cd_id INTEGER, cd_title TEXT, band_id INTEGER);
copy cd from stdin with delimiter as '&';
10&De Stijl&10
20&Elephant&10
30&Icky Thumb&10
30&Live At Leeds&20
40&The Spliff Radio Show&40
50&Check Your Head At The Door&50
60&Sticky Fingers&30
70&Let It Bleed&30
80&Get Yer Ya-Ya's Out!&30
\.

--' ###################################################################

-- hier muss die funktion geschrieben werden

-- ###################################################################



SELECT band_name, cd_title FROM
   band INNER JOIN cd ON band.band_id = cd.band_id;

SELECT band_name, cd_liste(band_id) from band;
```