# Spaß mit PostgreSQL

Peter Eisentraut

Chemnitzer Linux-Tage 2009

# Der Anfang allen Übels . . .

Procedural Languages:

- beliebige Programmiersprachen im PostgreSQL-Server ausführen
  - Perl, Python, Tcl, Ruby, Java, etc.
- für Anwendungslogik
- Anbindung an externe Bibliotheken

# Procedural Languages: Beispiel

```
CREATE OR REPLACE FUNCTION email_name(email text)
RETURNS text
LANGUAGE plperlu
AS $$
use Email::Address;

my @addresses = Email::Address->parse($_[0]);
return undef unless scalar(@addresses) > 0;
return $addresses[0]->name;
$$;
```

# Beispiel PL/sh

Code mit beliebigen Shells:

```
CREATE FUNCTION email(addr text, subj text,
                      body text)
RETURNS void
LANGUAGE plsh
AS $$
#!/bin/sh
echo "$3" | mail -s "$2" "$1"
$$;
```

# PL(/sh) Interna

- Quelltext kann beliebig verarbeitet werden, mit Bibliothek, Interpreter o. ä.
- PL/sh kopiert Quelltext in temporäre Datei
- Interpreter wird anhand `#!` ausgesucht
- Argument sind in $1, . . .
- Eingeschränkte Flexibilität bei Triggern

# PL/sh Demo

Benachrichtigungen über Updates senden

# PL Demo mit GUI

PL/PerI ist flexibiler bei der Bearbeitung von Triggern

# PL/LOLCODE

```
create or replace function lol_pi() returns float immutable
language pllolcode as $$
HAI
    I HAS A PIADD ITZ 0.0
    I HAS A PISUB ITZ 0.0
    I HAS A ITR ITZ 0
    I HAS A ITRZ ITZ 20000
    I HAS A T1
    I HAS A T2

    IM IN YR LOOP
        T1 R QUOSHUNT OF 4.0 AN SUM OF 3.0 AN ITR
        T2 R QUOSHUNT OF 4.0 AN SUM OF 5.0 AN ITR
        PISUB R SUM OF PISUB AN T1
        PIADD R SUM OF PIADD AN T2
        ITR R SUM OF ITR AN 4.0
        BOTH SAEM ITR AN BIGGR OF ITR AN ITRZ, O RLY?
            YA RLY, GTFO
        OIC
    IM OUTTA YR LOOP
    FOUND YR SUM OF 4.0 AN DIFF OF PIADD AN PISUB
KTHXBYE
$$;
```

# psql Customization

Initialisierungsdatei `~/.psqlrc` oder `/etc/psqlrc`

```
\set QUIET yes
\pset null _null_
\set HISTCONTROL ignoredups
\set VERBOSITY verbose
\set PROMPT1 '%[%033[1;32;40m%]%n@%m:%~%x%R%#%[%033[0m%] '
\set PROMPT2 :PROMPT1
```

# Rekursive Anfragen: Beispiel

```
WITH RECURSIVE Z(IX, IY, CX, CY, X, Y, I) AS (
    SELECT IX, IY, X::float, Y::float, X::float, Y::float, 0
        FROM (select -2.2 + 0.031 * i, i from generate_series(0,101) as i) as xgen(x,ix),
             (select -1.5 + 0.031 * i, i from generate_series(0,101) as i) as ygen(y,iy)
    UNION ALL
    SELECT IX, IY, CX, CY, X * X - Y * Y + CX AS X, Y * X * 2 + CY, I + 1
        FROM Z
        WHERE X * X + Y * Y < 16::float
              AND I < 100
)
SELECT array_to_string(array_agg(SUBSTRING(' .,,,-----++++%%%%@@@@#### ',
                                 LEAST(GREATEST(I,1),27), 1)),'')
FROM ( SELECT IX, IY, MAX(I) AS I FROM Z GROUP BY IY, IX ORDER BY IY, IX ) AS ZT
GROUP BY IY
ORDER BY IY;
```

# Rekursive Anfragen: Wie funktioniert's?

1. Nicht-rekursiven Teil ausführen, in Ergebnistabelle und Arbeitstabelle speichern
2. Wenn Arbeitstabelle nicht leer, dann ...
3. Rekursiven Teil ausführen, mit Arbeitstabelle anstelle Platzhalter, an Ergebnistabelle anhängen und in Zwischentabelle speichern
4. Zwischentabelle in Arbeitstabelle übertragen
5. wiederholen

# Rekursive Anfragen: Beispiel 2

```
CREATE TABLE teile (
   ganzes text,
   teil text,
   menge int
);

INSERT INTO teile VALUES ('Auto', 'Motor', 1),
                         ('Auto', 'Rad', 4),
                         ('Motor', 'Zylinderkopf', 1),
                         ('Rad', 'Schraube', 5),
                         ('Zylinderkopf', 'Schraube', 14);

WITH RECURSIVE enthaltene_teile(ganzes, teil, menge) AS (
    SELECT ganzes, teil, menge FROM teile WHERE ganzes = 'Auto'
  UNION ALL
    SELECT t.ganzes, t.teil, t.menge * et.menge
    FROM enthaltene_teile et, teile t
    WHERE t.ganzes = et.teil
  )
SELECT teil, SUM(menge) AS gesamtmenge
FROM enthaltene_teile
GROUP BY teil;
```

# Rekursive Anfragen: Beispiel 3

# Einbindung externer Daten

- andere PostgreSQL-Systeme abfragen
- andere DBMS-Typen abfragen
- Nicht-SQL-Daten lesen (CSV, XML)

# Externe Daten: Beispiel

DBI-Link: externe DBI-kompatible Datenquellen als PostgreSQL-Tabellen darstellen
z.B. MySQL über PostgreSQL

# Externe Daten: Beispiel 2

Also gehen auch beliebige Daten, z. B. DBD::CSV oder DBD::AnyData . . .

Nette Idee, geht aber nicht: Diese Treiber exportieren `table_info` und/oder `column_info` nicht.

# Externe Daten: Beispiel 2

Also gehen auch beliebige Daten, z. B. DBD::CSV oder DBD::AnyData . . .
Nette Idee, geht aber nicht: Diese Treiber exportieren `table_info` und/oder `column_info` nicht.

# Externe Daten: SQL/MED

“Management of External Data”

- Teil des SQL-Standards
- Implementierung geplant ab PostgreSQL 8.5
- vereinheitlicht diverse bestehende Ad-Hoc-Lösungen
- evtl. auch Unterstützung für abenteuerliche Quellformate

(Teil-)Beispiel:

```
CREATE FOREIGN TABLE data
  SERVER extradb
  OPTIONS (tablename 'DATA123');
```

# Web Services

Beispiel Google Translate

# PostgreSQL und Mobiltelefone

Probleme:

- RAM
- Flash-Speicher
- POSIX-API
- CPU mit Spinlocks
- (Geschwindigkeit?)

PostgreSQL-Server auf Handy geht wohl (noch) nicht.

# PostgreSQL GUIs und Mobiltelefone

- psql nativ nicht portiert
- psql über ssh geht gut
- phpPgAdmin geht gut
- pgAdmin III geht (noch) nicht
- pgPhoneHome

# Externe Daten umgekehrt

Wie administrieren?

- psql ist zu spartanisch
- pgAdmin ist Mist :-/
- Erst wollte ich einen kioslave schreiben . . .
- mc ist am besten
- . . . aber Plugin schreiben wollte ich nicht.

# PostgreSQL und FUSE

# Zusammenfassung

- Mit . . .
- vielen Programmiersprachen,
- vielen Bibliotheken,
- vielen Datenquelllen,
- vielen Schnittstellen
- . . . ist ’ne Menge Unsinn möglich.
- (Aber nicht aller funktioniert.)