Какво ново в PostgreSQL 9.3

Валентин Черноземски

valentin.chernozemski@gmail.com

skype @ valmilski

http://www.siteground.com

http://www.1h.com

@OpenFest 2013

PostgreSQL - кратко описание

- Лесна за конфигуриране и администриране
- Подробно документирана
- Бърза
- Безплатна
- Стабилна
- Многофункционална
- Огромно общество
- С познат синтаксис
- Даваща огромна свобода за интерпретация на вашите нужди
- Weapon of choice

Бързо & бързо && още по-бързо

- Стремеж към бързодействие с минимални ресурси
- Намаляване на разходите
- Опростяване и ускоряване на изпълнение на задачите
- Бързото получаване на резултати – A.S.A.P :D
- При добро познаване на нуждите ни, можем лесно да получим неколкократно бързодействие с минимални усилия … или не :D
- За целта трябва да познаваме, както данните с които боравим така и инструментите с които ги интерпретираме
- Резултат → множество нововъведения

NO KEY UPDATE – Масови ъпдейти на колони без блокиране

При условие че не ъпдейтваме полето към което има foreign key (Ключ)

Ключ	Име	Пол
1	Иванчо	Мъж
2	Мариика	Жена

Foreign key t1(Ключ)	Програма	Предаване
1	К1	Лов и Риболов
1	К2	Шоуто на Пъци
2	К1	Сериалисимо
2	К2	Реклами на всеки 30 мин.

- Как

  UPDATE t1

  set Пол = 'Неопределен'

  NO KEY UPDATE

- **Вижте още за проблема на:**

  http://momjian.us/main/writings/pgsql/features.pdf

  http://mina.naguib.ca/blog/2010/11/22/postgresql-foreign-key-deadlocks.html

  http://goo.gl/n8gaQz

GiST индексите могат да бъдат unlogged

- GiST индекс - Базов метод с чиято помощ могат да се имплементират различни схеми за индексиране според структурата на данните с които боравим с цел бързодействие
- GiST индексите могат да бъдат unlogged
- Unlogged? - От PostgreSQL 9.1

  Данните които се записват в unlogged таблици/индекси:
- Не се записват в write-ahead (wal) лог-а - по малко операции за запис в/у диска
- Не се репликират към “slave” PostgreSQL сървърите
- Това ги прави по бързи
- Не се препоръчват за критични данни! Данните в unlogged таблиците се изчистват **напълно** при стартиране след crash
- Всички индекси направени върху unlogged таблици също са unlogged
- Много добър избор когато оптимизираме таблици с незначителни данни но с много голяма степен на запис върху тях

Повишена ефективност при използване на commit_delay

- Подобрена синхронизация на сесиите чакащи да изтече commit_delay интервала
- commit_delay позволява да се намали броят на произволните записи на диска
- Групира сходните записите на много малки промени в една единствена транзакция
- Не чака ако няма сходни транзакции
- Не се взима в предвид ако fsync е изключен

Увеличена скорост на доставяне на промените към standby сървъра при репликация с изключен synchronous_commit

- synchronous_commit = off е безопасно от гледна точка на цялостност на самата база данни
- При изключване на тази опция губим записите извършвали се по време на crash но базата остава в работещо състояние
- Може да се използва за намаляване на сроковете на доставка на промените към standby сървърите
- Жертваме сигурност на записите но не и DB corruption в случай на crash
- Намалява натоварването от записи върху диска
- Трябва да се използва внимателно според важността на данните с които боравим

Докато мигнеш и аз вече съм “шеф”

- Позволява на standby сървъра да стане главен за изключително кратко време
- Няма достатъчно документация :(
- Allows replicas to be promoted in less than a second, permitting 99.999% uptime. More details TBD.
- Sub-second "fast failover" option when switching replicated servers, for 99.999% high availability.

Паралелно извличане на данните в текстов SQL формат

- pg_dump –jobs='X'/-j X позволява да извличаме няколко таблици едновременно
- Намалява времето за правене на резервни копия в SQL формат
- Увеличава натоварването за четене върху диска и върху самата база
- Копията на таблиците се записват в отделни файлове в определена директория
- Тестове (крадени :))

- Размер: 4.5GB
- Компресиран: 370MB
- (no -j): 1m3s
- -j2: 0m28s
- -j3: 0m24s
- -j4: 0m24s
- -j5: 0m25s

Аха! И други благинки!

- Възможност за правене на индекси за бързо търсене с регулярни изрази - pg_trgm
- Вече можем да окажем на сървъра да слуша на повече от един сокет (chroot и др.) с помоща на unix_socket_directories

Аха! И други благинки!

- Възможност за правене на индекси за бързо търсене с регулярни изрази - pg_trgm
- Вече можем да окажем на сървъра да слуша на повече от един сокет (chroot и др.) с помоща на unix_socket_directories

Сега пикантно от кухнята

- PostgreSQL foreign data wrappers (FDW) – обвивка (подправка) за мешана скара
- PostgreSQL streaming replication timeline switch
- Работа с JSON + PostgreSQL

Специалитетите на кухнята …
PostgreSQL **Writable** FDW (foreign data wrappers)

- Що е то

  Достъп до разнородни ресурси от данни

  Management of external data – 8.4

  FDW extensions – 9.1

  Айде давай по същество че ми е интересно (спи ми се!)

PostgreSQL + Redis FDW - Как

- **Инсталация**

  yum install postgresql93-devel

  cd /usr/src

  git clone https://github.com/pg-redis-fdw/redis_fdw

  cd redis_fwd

  git checkout REL9_3_STABLE

  PATH=/usr/pgsql-9.3/bin:$PATH make USE_PGXS=1 install

- **В PostgreSQL**

  PostgreSQL=# CREATE EXTENSION redis_fdw;

  PostgreSQL=# \dx redis_fdw;

  Name | Version | Schema | Description

  -----------+---------+--------+--

  redis_fdw | 1.0 | public | Foreign data wrapper for querying a Redis server

- **Работи**

И навързахме нещата

- **Свързване на PostgreSQL към Redis**

  PostgreSQL=# CREATE SERVER redis_server FOREIGN DATA WRAPPER redis_fdw OPTIONS (address '127.0.0.1', port '6379');

  CREATE SERVER

  PostgreSQL=# CREATE FOREIGN TABLE redis_db0 (key text, value text) SERVER redis_server OPTIONS (database '0');

  CREATE FOREIGN TABLE

  PostgreSQL=# CREATE USER MAPPING FOR PUBLIC SERVER redis_server OPTIONS (password '');

- **Добавяне на няколко редис записа през redis-cli**

  [root@pgm redis_fdw]# redis-cli

  rredis 127.0.0.1:6379> set openfest2012 cool

  OK

  rredis 127.0.0.1:6379> set openfest2013 awesome

  OK

  redis 127.0.0.1:6379>

- **Проверка дали записите се виждат през PostgreSQL**

  PostgreSQL=# SELECT * FROM redis_db0;

  ```
      key      |  value
  --------------+---------
   openfest2012 | cool
   openfest2013 | awesome
  ```

I failed … miserably!

- Промяна на стойности в Redis през PostgreSQL

  PostgreSQL=# update redis_db0 set value='uber-turbo-geeky-awesome-dudeee';

  ERROR: cannot update foreign table "redis_db0"

- Оказа се че redis wrapper-а още не е преправен да поддържа записи

  redis 127.0.0.1:6379> del openfest2013

  (integer) 1

  redis 127.0.0.1:6379> set openfest2013 **epic-writable-fail**

  OK

  PostgreSQL=# SELECT * FROM redis_db0;

  key | value

  ----------+--------------------

  openfest2012 | cool

  openfest2013 | **epic-writable-fail**

- **Поне да проверим дали join между PostgreSQL и Redis работят**

Let them join, let them join, let them join ...

- **Създаване на тестова таблица**

  PostgreSQL=# CREATE TABLE psql_db (psql_key text, psql_value text);

  CREATE TABLE

- **Добавяне на записи за тестове**

  PostgreSQL=# INSERT INTO psql_db (psql_key, psql_value) VALUES ('openfest2013-psql', 'epic-writable-fail');

  INSERT 0 1

- **Проверка дали можем да корелираме данните**

  PostgreSQL=# SELECT p.psql_key, r.value FROM psql_db p, redis_db0 r WHERE p.psql_value=r.value and r.value = 'epic-writable-fail';

  psql_key | value

  ------------------+--------------------

  openfest2013-psql | epic-writable-fail

- **Изглежда че работи :)**

Лъжец! Уж можехме да пишем в чужди бази и да ходим по жени? Все още можете :) **PostgreSQL && Kyoto Tycoon**

- **Инсталиране на Kyoto Tycoon**

  cd /usr/src && wget http://fallabs.com/kyototycoon/pkg/kyototycoon-0.9.56.tar.gz

  tar xzf kyototycoon-0.9.56.tar.gz

  cd kyototycoon-0.9.56

  yum install kyotocabinet-devel gcc-c++ lua.x86_64 lua-devel.x86_64 lua-inotify.x86_64

  ./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var --libdir=/usr/lib64 --enable-devel –enable-lua && make -j4 && make check && make install

- **Инсталиране на Kyoto Tycoon FDW**

  git clone https://github.com/cloudflare/kt_fdw

  cd kt_fdw/

  make && make install (required editing of Makefile first)

- **Пускане на Kyoto Tycoon DB сървъра**

  ktserver -host 127.0.0.1 -scr /usr/src/kt_fdw/transactions.lua

- **Създаване на extension-а в PostgreSQL**

  PostgreSQL=# CREATE EXTENSION kt_fdw;

  CREATE EXTENSION

Свързване на PostgreSQL към KT

- **Задаване на FDW сървър с име kt_srv**

  PostgreSQL=# CREATE SERVER kt_srv FOREIGN DATA WRAPPER kt_fdw;

  CREATE SERVER

  PostgreSQL=# CREATE USER MAPPING FOR PUBLIC SERVER kt_srv;

  CREATE USER MAPPING

- **Създаване на таблица директно в Kyoto Tycoon сървъра през PostgreSQL**

  PostgreSQL=# CREATE FOREIGN TABLE kt_table (kt_key TEXT, kt_value TEXT) SERVER kt_srv;

  CREATE FOREIGN TABLE

- **Добавяне на запис в Kyoto Tycoon през PostgreSQL**

  PostgreSQL=# INSERT INTO kt_table (kt_key, kt_value) VALUES ('openfest2013-kt', 'epic-writable-fail');

  INSERT 0 1

- **Извличане на записа**

  PostgreSQL=# select * from kt_table ;

  ```
      kt_key      |      kt_value
  ----------------+--------------------
   openfest2013-kt | epic-writable-fail
  ```

Достъп до данните през различните интерфейси/приложения

- **Проверка дали записа който добавихме през PostgreSQL е видим за KT базата с помоща на ktremotemgr**

  ktremotemgr list -host 127.0.0.1

  openfest2013-kt

  ktremotemgr get -host 127.0.0.1 openfest2013-kt

  epic-writable-fail

- **Промяна на записа през KT**

  ktremotemgr set -host 127.0.0.1 openfest2013-kt epic-writable-win

- **Проверка на промяната в PostgreSQL**

  PostgreSQL=# select * from kt_table ;

  kt_key | kt_value

  ----------+----------

  openfest2013-kt | epic-writable-win

- **Последен join – сбогом любима**

  PostgreSQL=# SELECT p.psql_key, r.key, k.kt_key, r.value FROM psql_db p, redis_db0 r, kt_table k WHERE p.psql_value=r.value and k.kt_value=r.value and r.value = 'epic-writable-fail';

  psql_key | key | kt_key | value

  ----------+----------+----------+----------

  openfest2013-psql | openfest2013 | openfest2013-kt | **epic-writable-win**

Какво още за FDW

- Защо би ми притрябвало да ползвам FDW
- С подходящия драйвер всеки (?) източник на информация може да се превърне в база

  text

  csv

  twitter

  anything … you just **code** it :)

PostgreSQL replication - timeline ключа

- Постоянна достъпност
- Балансиране на заявките за четене
- Възстановяване от катаклизми (can't handle truncate/drop table/db)
- Безболезнен failover
- Главният проблем: frustration free slave re-synchronization без доставка на wal logs

5 мин. Конфигурация на master/slave в PostgreSQL 9.3

- **Първо настройваме мастера**
- **Генериране на SSL ключове в /var/lib/pgsql/9.3/data**

  server.crt

  server.key

  server.req
- **/var/lib/pgsql/9.3/data/postgresql.conf**

  listen_addresses = '*'

  wal_level = hot_standby

  max_wal_senders = 3

  hot_standby = on

  ssl = on

  ssl_ciphers = 'ALL:!ADH:!LOW:!EXP:!MD5:@STRENGTH'
- **Fix /var/lib/pgsql/.pgpass && chown postgres: /var/lib/pgsql/.pgpass && chmod 600 /var/lib/pgsql/.pgpass**
- **/var/lib/pgsql/9.3/data/pg_hba.conf**
- local replication postgres md5
- host replication postgres 127.0.0.1/32 m5
- host replication postgres master.ip.ip.ip.ip/32 md5
- host replication postgres slave1.ip.ip.ip.ip/32 md5
- host replication postgres slave2ip.ip.ip.ip/32 md5

Инициализиране и стартиране на подчинените

- **Стартираме сървъра и правим копие което ще дистрибутираме на slave сървърите**

  pg_basebackup -D pg_backup -R -X stream -Upostgres -d 'user=postgres port=5432 sslmode=require sslcompression=1 host=master.ip.ip.ip'

- **НОВО:** В pg_backup/recovery.conf добавяме следните 2 реда

  trigger_file = '**/tmp/finish.replication**'

  recovery_target_timeline = 'latest'

- **Копираме backup-а на слейв сървърите**

  rsync -aHzv -e "ssh -p22" pg_backup/ slave1.ip.ip.ip:/var/lib/pgsql/9.3/data/

  ssh slave1.ip.ip.ip 'chown postgres: /var/lib/pgsql/9.3/data/ && /etc/init.d/postgresql-9.3 start'

- **Проверка на слейв машините /var/lib/pgsql/9.3/data/pg_log/postgresql-*.log**

  < 2013-11-01 15:29:00.530 CDT >LOG: database system is ready to accept read only connections

  < 2013-11-01 15:29:00.574 CDT >LOG: started streaming WAL from primary at 0/A000000 on timeline 1

Тестове master/slave

- **Master**

  template1=# CREATE TABLE replica_test (name text, age text);

  CREATE TABLE

  template1=# insert into replica_test VALUES ('vaL', '28');

  INSERT 0 1
- **Slave1 && Slave2**

  template1=# select * from replica_test;

  name | age

  ------+-----

  vaL | 28

  (1 row)

  template1=#

The timeline switch test

- **Спиране на мастера**

  [root@pgm ~]# /etc/init.d/postgresql-9.3 stop

  Stopping postgresql-9.3 service: [OK]

- **Копията още работят (readonly)**

  template1=# select * from replica_test;

   name | age

  ------+-----

   vaL | 28

- **Гледаме лога**

  < 2013-11-01 15:53:33.382 CDT >FATAL: could not connect to the primary server: could not connect to server: Connection refused Is the server running on host "master.ip.ip.ip" and accepting TCP/IP connections on port 5432?

- **Инструктираме pgslave1 да стане шеф**

  touch /tmp/finish.replication

- **В лога се вижда как бившият pgslave1 вече е мастер**

  < 2013-11-01 15:53:38.389 CDT >LOG: trigger file found: /tmp/finish.replication

  < 2013-11-01 15:53:38.389 CDT >LOG: redo done at 0/A01F968

  < 2013-11-01 15:53:38.389 CDT >LOG: last completed transaction was at log time 2013-11-01 15:48:37.237248-05

  < 2013-11-01 15:53:38.389 CDT >LOG: selected new timeline ID: 2

  < 2013-11-01 15:53:38.550 CDT >LOG: archive recovery complete

  < 2013-11-01 15:53:38.585 CDT >LOG: database system is ready to accept connections

  < 2013-11-01 15:53:38.585 CDT >LOG: autovacuum launcher started

Timeline ключа и да видим как се справя с теста

- На **slave1** добавяме нов запис

  template1=# insert into replica_test VALUES ('Ski', '28');

  INSERT 0 1

- През това време **slave2** още се опитва да се свърже към стария **master** и той не вижда този запис
- На **slave2** преконфигурираме /var/lib/pgsql/9.3/data/recovery.conf така че да се свърже към **IP-то** на новия мастер **slave1**
- **Рестартираме PostgreSQL на slave2**

  /etc/init.d/postgresql restart

Резултатите от теста

- **Преди рестарта**

  < 2013-11-01 15:55:13.907 CDT >FATAL: could not connect to the primary server: could not connect to server: Connection refused
  Is the server running on host "184.154.13.202" and accepting
  TCP/IP connections on port 5432?

- **По време на и след рестарта**

  < 2013-11-01 15:55:15.965 CDT >LOG: received fast shutdown request
  < 2013-11-01 15:55:15.965 CDT >LOG: aborting any active transactions
  < 2013-11-01 15:55:15.967 CDT >LOG: shutting down
  < 2013-11-01 15:55:15.969 CDT >LOG: database system is shut down
  < 2013-11-01 15:55:17.056 CDT >LOG: database system was shut down in recovery at 2013-11-01 15:55:15 CDT
  < 2013-11-01 15:55:17.057 CDT >LOG: **entering standby mode**
  < 2013-11-01 15:55:17.059 CDT >LOG: consistent recovery state reached at **0/A01F9D0**
  < 2013-11-01 15:55:17.059 CDT >LOG: record with zero length at **0/A01F9D0**
  < 2013-11-01 15:55:17.060 CDT >LOG: database system is ready to accept read only connections
  < 2013-11-01 15:55:17.068 CDT >LOG: **fetching timeline history file for timeline 2 from primary server**
  < 2013-11-01 15:55:17.077 CDT >LOG: started streaming WAL from primary at 0/A000000 on timeline 1
  < 2013-11-01 15:55:17.084 CDT >LOG: **replication terminated by primary server**
  < 2013-11-01 15:55:17.084 CDT >DETAIL: End of WAL reached on timeline 1 at 0/A01F9D0.
  < 2013-11-01 15:55:17.085 CDT >LOG: **new target timeline is 2**
  < 2013-11-01 15:55:17.086 CDT >LOG: **restarted WAL streaming at 0/A000000 on timeline 2**
  < 2013-11-01 15:55:17.148 CDT >LOG: **redo starts at 0/A01F9D0**

PostgreSQL 9.3 + JSON

- Какво е JSON и къде можем да го използваме
- Sick of joins
- Ограничения (1GB – JSON се пази като текст)
- Базата се грижи за валидацията на формата при запис

Примерен проблем
Опис на домакинства

- Домакинство
  - Членове
    - Име
      - Възраст
      - Пол
      - Хобита
        - Риболов
        - Плетене
      - Музикални предпочитания
        - Рок
        - Кънтри

Примерни данни

```
my $data = {
    members =>  {
        'Ivancho'   =>  {
            age =>  18,
            hobbyes =>  ['Sport', 'Fishing'],
            music   =>  {
                Rock    =>  {
                    Titles  =>  ['Stairway To Heaven', "Sweet Child O' Mine"],
                    Artists =>  ['Bob Seger', 'Led Zeppelin']
                },
                Pop     =>  {
                    Titles  =>  ['Song A', 'Song B'],
                    Artists =>  ['Artist X', 'Artist Y']
                }
            }
        },
        'Mariika'   =>  {
            age =>  21,
            hobbyes =>  ['Kiflingб', 'Golddigging'],
            music   =>  {
                Chalga  =>  {
                    Titles  =>  ['Hvani me', 'Liubime'],
                    Artists =>  ['Haivana', 'Mondio']
                },
                Maaneta     =>  {
                    Titles  =>  ['Hasmi Giubek', 'Bash bash kiuchek'],
                    Artists =>  ['Sulio', 'Pulio']
                }
            }
        }
    }
};
```

Данни сериализирани с помоща на JSON … my eyes are bleeding too

- {"members":{"Ivancho":{"music":{"Rock":{"Titles":["Stairway To Heaven","Sweet Child O' Mine"],"Artists":["Bob Seger","Led Zeppelin"]},"Pop":{"Titles":["Song A","Song B"],"Artists":["Artist X","Artist Y"]}},"hobbies":["Sport","Fishing"],"age":18},"Mariika":{"music":{"Chalga":{"Titles":["Hvani me","Liubime"],"Artists":["Haivana","Mondio"]},"Maaneta":{"Titles":["Hasmi Giubek","Bash bash kiuchek"],"Artists":["Sulio","Pulio"]}},"hobbies":["Kifling","Golddigging"],"age":21}}}

Добавяне на записи

- CREATE TABLE households (id serial, hh_data **json**);
- postgres=# INSERT INTO households (hh_data) VALUES ('{ГОЛЯМОТО КИФТЕ ОТ ПРЕДНИЯТ СЛАЙД}');
- template1=# select * from households;
- 1 | {"members":{"Ivancho1":{"music":{
- 2 | {"members":{"Ivancho2":{"music":{
- 3 | {"members":{"Ivancho3":{"music":{
- (3 rows)
- template1=#
-

Оператори за работа с JSON

- **row_to_json**

  SELECT **row_to_json**(row(buy,sell,description,defense))

  {"f1":200,"f2":80,"f3":"basic shield","f4":7}

- **array_to_json**

  SELECT **row_to_json**(row(**array_to_json**(fields), description)) FROM rpg_items_attack;

  {"f1":[500,200,10],"f2":"basic sword"}

- -> && ->>

  template1=# select hh_data->'members'->'Ivancho'->'music'->'Rock'->'Artists' from households where id=1;

  ["Bob Seger","Led Zeppelin"]

- #> && #>>

  template1=# select hh_data->'members'->'Ivancho'->'music'->'Rock'#>'{Artists,1}' from households where id=1;

  "Led Zeppelin"

Функции за извличане на json

- **json_each – key/value формат**

  template1=# SELECT * FROM json_each((SELECT hh_data->'members'->'Ivancho'->'music' FROM households WHERE id = 1));

  Rock | {"Titles":["Stairway To Heaven","Sweet Child O Mine"],"Artists":["Bob Seger","Led Zeppelin"]}

  Pop | {"Titles":["Song A","Song B"],"Artists":["Artist X","Artist Y"]}

- **json_extract_path just like -> && ->>**

  template1=# SELECT **json_extract_path**(hh_data, 'members', 'Ivancho', 'music') FROM households WHERE id=1;

  {"Rock":{"Titles":["Stairway To Heaven","Sweet Child O Mine"],"Artists":["Bob Seger","Led Zeppelin"]},"Pop":{"Titles":["Song A","Song B"],"Artists":["Artist X","Artist Y"]}}

- **json_object_keys – извличане на ключове**

  template1=# SELECT **json_object_keys**(hh_data->'members') FROM households WHERE id=1;

  Ivancho

  Mariika

JSON новостите в 9.3

- Добавяне на нови функции и оператори за извличане на елементи от JSON (Andrew Dunstan)
- Възможност JSON записите да бъдат конвертирани в нормални записи (Andrew Dunstan)
- Въведени функции за конвертиране на scalar, records и hstore стойности в JSON (Andrew Dunstan)

Препратки

- **About**
- http://www.postgresql.org/about/
- http://www.postgresql.org/about/licence/
- http://www.postgresql.org/docs/9.3/static/
- http://en.wikipedia.org/wiki/ACID
- **What's new**
- http://www.postgresql.org/docs/9.3/static/release-9-3.html
- http://www.postgresql.org/docs/9.3/static/pgtrgm.html
- http://lwn.net/Articles/550418/
- http://wiki.postgresql.org/wiki/What%27s_new_in_PostgreSQL_9.3
- **Writable FDW**
- http://www.slideshare.net/amdunstan/pgcon-redis
- http://wiki.postgresql.org/wiki/Foreign_data_wrappers
- http://fallabs.com/kyototycoon/spex.html
- https://github.com/cloudflare/kt_fdw
- https://github.com/pg-redis-fdw/redis_fdw
- **Replication Timeline Switching**
- http://www.depesz.com/2012/12/22/waiting-for-9-3-allow-a-streaming-replication-standby-to-follow-a-timeline-switch/
- http://www.postgresql.org/docs/9.3/static/recovery-target-settings.html
- **JSON**
- **http://michael.otacoo.com/postgresql-2/postgres-9-2-highlight-json-data-type/**
- **http://michael.otacoo.com/postgresql-2/postgres-9-3-feature-highlight-json-data-generation/**
- **http://michael.otacoo.com/postgresql-2/postgres-9-3-feature-highlight-json-operators/**
- **http://michael.otacoo.com/postgresql-2/postgres-9-3-feature-highlight-json-parsing-functions/**
- **http://www.postgresql.org/docs/9.3/static/functions-json.html**
- **https://speakerdeck.com/selenamarie/schema-liberation-with-json-and-plv8-and-postgres**
- **https://postgres.heroku.com/blog/past/2013/6/5/javascript_in_your_postgres/**
-

___EOF___

Въпроси?