Python dentro de Postgres con PL/Python

PGDAY Ecuador 2011
PUCE – Quito
Milton Labanda
Octubre 2011

Stored Procedures

- Funciones o procedimientos persistentes dentro de la Base de Datos
- Se necesita un Lenguaje Procedural (PL) : Un binding o una pasarela hacia el entorno real de ejecución del lenguaje escogido
- Se puede usar todas las posibilidades del lenguaje escogido en el procedimiento almacenado
- Muy útiles para extender las posibilidades de nuestra Base de Datos

Stored Procedures: funciones en Postgres

- Varias alternativas para escribir funciones almacenadas en Postgres :
  - PL/PgSql
  - PL/TCL
  - PL/Java
  - PL/Perl
  - PL/Python

Por qué Python?

- Fácil de Leer, fácil de aprender, fácil de usar
- Poderoso y rápido
- Disponible en casi todas las distribuciones GNU/Linux, también hay para Window$
- Cientos de módulos estándar y de terceros
- Ej: Gnome, OpenERP, Google, Yahoo, PDVSA ...
- Un futuro muy prometedor

plpythonu: Por qué la 'u' de untrusted?

- PL/Python es un lenguage UNTRUSTED
- Sólo los superusuarios de la Base de Datos pueden crear funciones
- Las funciones pueden comunicarse fuera de la Base de Datos

Requerimientos

- Instalar **PostgreSQL**
- Instalar **Python**
- Instalar **postgresql-python**:
  - apt-get install postgresql-python (en debian)
- Habilitar PL/Python en la Base de Datos:
  - 1) desde dentro:
    - bd=# CREATE LANGUAGE plpythonu;
  - 2) desde fuera
    - $ createlang plpythonu nombre_bd

Sintaxis para crear funciones


```
CREATE FUNCTION nombre_funcion (....)
  AS $$
...código python...
 $$ LANGUAGE plpythonu;
```

Una primera función


```
CREATE OR REPLACE FUNCTION
potenciapy(base float, exponente float)
RETURNS float AS
$$
    p = base ** exponente
    return p
$$
LANGUAGE 'plpythonu' VOLATILE;
```

Una primera función: resultados


```
SELECT potenciapy(2,4)

potencia
- - - - - - -
     16
```

Mapeo de Tipos de Datos

PostgreSQL	Python	NOTA
boolean	bool	
Smallint, int	int	
Real, double, numeric	float	
otros	str	
arrays	listas	
Composite Types	diccionarios	Como parámetros
Composite Types	tuplas, listas, diccionarios, objectos de clases Python	Como valores de retorno

Tipos Compuestos: tipo Postgres


```
CREATE TYPE persona AS (
  nombre   TEXT,
  apellido TEXT
);
```

Tipos Compuestos: clase Python


```
CREATE FUNCTION crear_persona (nombre TEXT,
apellido TEXT)  RETURNS persona AS $$
    class Persona:
        def __init__ (self, n, a):
        self.nombre = n
        self.apellido = a
    return Persona(nombre, apellido)
$$ LANGUAGE plpythonu;
```

Tipos Compuestos: acción...


```
db=# SELECT crear_persona(
'Guido', 'van Rossum'
);
    crear_persona
----------------------
 (Guido,"van Rossum")
```

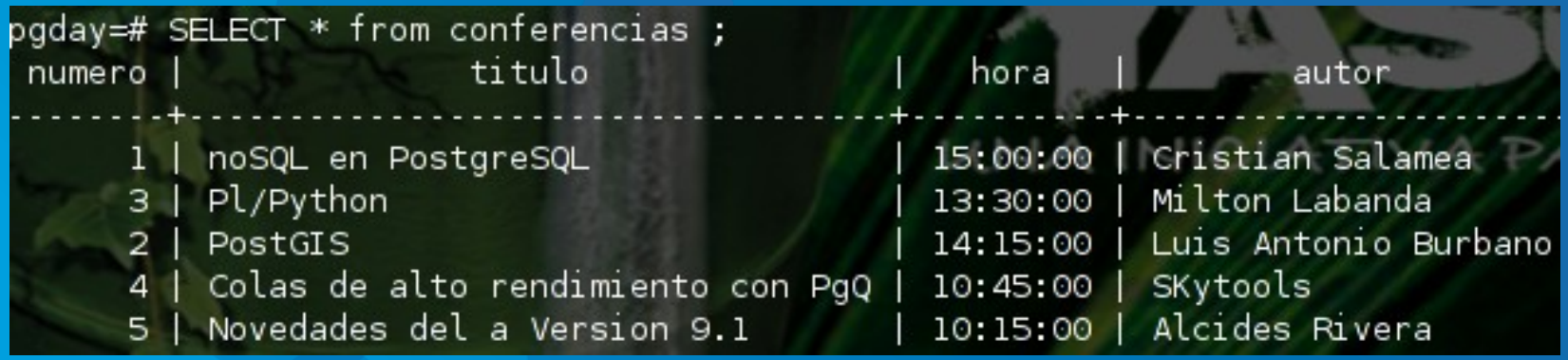
setof: Múltiples resultados


```
create or replace function pydir(dir text)
returns setof text
as $$
    import os
    return os.listdir(dir)
$$ language 'plpythonu';
```

setof: Múltiples resultados


```
db=# select pydir('/etc/postgresql/9.0/main');
      pydir
-----------------
 pg_ident.conf
 start.conf
 environment
 pg_hba.conf
 pg_ctl.conf
 postgresql.conf
```

Una BD para los ejemplos que siguen



```
pgday=# SELECT * from conferencias ;
 numero |                titulo                |   hora   |        autor
--------+--------------------------------------+----------+----------------------
      1 | noSQL en PostgreSQL                  | 15:00:00 | Cristian Salamea
      3 | Pl/Python                            | 13:30:00 | Milton Labanda
      2 | PostGIS                              | 14:15:00 | Luis Antonio Burbano
      4 | Colas de alto rendimiento con PgQ    | 10:45:00 | SKytools
      5 | Novedades del a Version 9.1          | 10:15:00 | Alcides Rivera
```

plpy: Acceso a Bases de Datos

- PL/Python importa automáticamente el módulo **plpy**
- El módulo plpy contiene dos métodos:
  - plpy.execute(consulta,limite)
  - plpy.prepare(consulta,tipos)
- Los resultados emulan listas o diccionarios de python

plpy: Acceso a Bases de Datos


```
create or replace function json(query text)
       returns setof text
as $$
    rs = plpy.execute(query)
    # devuelve una lista de diccionarios
   return rs
$$ language 'plpythonu';
```

plpy: Acceso a Bases de Datos


```
db=# select json('select titulo,hora
from conferencias order by hora');

                                    json
-----------------------------------------------------
 {'hora': '13:30:00', 'titulo': 'Pl/Python'}
 {'hora': '14:15:00', 'titulo': 'PostGIS'}
 {'hora': '15:00:00', 'titulo': 'noSQL en PostgreSQL'}
```

DO: Bloques de Código Anónimos

```
db=# do $$
import os
dir = os.getcwd()
plpy.info('El directorio actual es %s' % dir)
$$ LANGUAGE plpythonu;
INFO:  El directorio actual es /home/miltonlab
CONTEXTO:  PL/Python anonymous code block
DO
```

Triggers y plpython: La función


```
create function pyaudit() returns trigger as $$
        import datetime
        t = datetime.datetime.now().__str__()
        d = (t,TD['table_name'],TD['event'],TD['when'])
        s = '%s tabla: %s evento: %s momento: %s' % d
        f = open('/tmp/pypglog.txt','a')
        f.write(tex+'\n')
        f.close()
$$ language plpythonu;
```

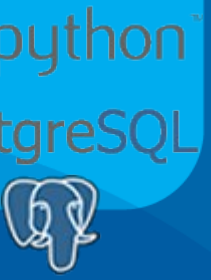
Triggers y plpython: el disparador


```
CREATE TRIGGER tr_confs
    BEFORE INSERT OR UPDATE OR DELETE
    ON conferencias
    FOR EACH ROW
   EXECUTE PROCEDURE pyaudit();
```

Triggers y plpython: acción...

```
db=# insert into conferencias values (
    4,'Colas con PgQ','10:45','Skytools'
);

$less /tmp/pypglog.txt
2011-10-02 08:12:05.115315 tabla: conferencias
evento: INSERT momento: BEFORE
2011-10-02 08:14:40.928435 tabla: conferencias
evento: INSERT momento: BEFORE
```



Preguntas?