PL/Python - Python in PostgreSQL

Stored procedures

- subroutine available to applications that access RDBMS
- callable from SQL
- more complicated tasks

Stored procedures - example

```
create function pystok(event_nr int) returns varchar AS
$$
begin
   return 'Pystok - edycja ' || event_nr::varchar;
end
$$ language plpgsql;
```

Stored procedures - example

```
pystok=# select pystok(23);

        pystok
-------------------
 Pystok - edycja 23
(1 wiersz)
```

Stored procedures

- many languages
- official support for PL/Perl, PL/Python, PL/Tcl, PL/pgSQL
- unofficial: Ruby, Java, R, V8, PHP

PL/Python

- untrusted
- available Python 2 and 3
- default Python 2

PL/Python

- Python Interpreter is initialised inside backend process
- Postgres types are transformed into Python types
- additional modules and variables are initialised

PL/Python

```
CREATE FUNCTION funcname (argument-list)
 RETURNS return-type
AS $$
 # PL/Python function body
$$ LANGUAGE plpythonu;
```

- BODY -> Python code
- get arg-list, return value
- if not returned - function returns None

PL/Python - say Hello

```
create function pymax (a integer, b integer)
 returns integer
as $$
 return a if a>b else b
$$ language plpythonu;
```

PL/Python

```
def __plpython_procedure_pymax_23456():
    return a if a>b else b
```

- 23456 - OID

PL/Python

```
def __plpython_procedure_pymax_23456():
    return a if a>b else b
```

- 23456 - OID
- arguments are set as global variables
- tip of the day: treat function parameters as read-only

PL/Python

- each function gets its own execution environment in the Python interpreter
- global data and function arguments from myfunc are not available to myfunc2
- exception - global dictionary GD

PL/Python - example

```
create or replace function pystok(event_nr integer) returns varchar as
$$
 return 'Pystok edycja {}'.format(event_nr)
$$ language plpythonu;
```

```
pystok=# select pystok(23);

        pystok
------------------
 Pystok edycja 23
(1 wiersz)
```

PL/Python - Data Types Mapping

PostgreSQL	Python
smallint, int	int
bigint, oid	long (Python 2), int (Python 3)
real	float
numeric	Decimal (cdecimal lub decimal)
varchar, other str types	str (Python 2 PostgresSQL encoding, Python 3 Unicode)

PL/Python - Composite Types

```
create table pystok (
 edition_nr integer,
 beer_quantity integer,
 good_speakers boolean
);
```

```
create or replace function is_good_party(edition pystok)
 returns boolean
as $$
if edition['beer_quantity'] > 50 and edition['good_speakers']:
   return True
return False
$$ language plpythonu;
```

PL/Python - Composite Types

```
pystok=# insert into pystok (edition_nr, beer_quantity,
good_speakers) values (23, 100, True);

pystok=# select is_good_party(pystok) happy from pystok;

 happy
-------
 t
(1 wiersz)
```

PL/Python - Triggers

- automatically executed or fired when some events occur
- e.g. UPDATE on table
- benefits:
  - event logging
  - generating some derived column values automatically
  - validate data

PL/Python - Triggers

- When a function is used as a trigger, the dictionary TD contains trigger-related values

TD[‘event’]	INSERT, UPDATE, DELETE, TRUNCATE
TD[‘when’]	BEFORE, AFTER, INSTEAD OF
TD[‘new’], TD[‘old’]	new/old values (depending on triggered event)
and more…	

PL/Python - Triggers

```
create or replace function check_active()
  returns trigger as $$
  class UserNotActive(Exception): pass

  if not TD["old"]["active"] and not TD["new"]["active"]:
     raise UserNotActive
$$ language plpythonu;
```

PL/Python - Triggers

```
create trigger change_username
before update on profile
for each row execute procedure
check_active();

insert into profile (username, active) values
('Marik1234', False);
```

PL/Python - Triggers

```
pystok=# UPDATE profile SET username='TheMarik1234'
WHERE id=1;

ERROR:  UserNotActive:
KONTEKST:  Traceback (most recent call last):
  PL/Python function "check_active", line 5, in <module>
      raise UserNotActive
PL/Python function "check_active"
```

PL/Python - Python modules

```
CREATE FUNCTION get_file_ext(file_path varchar)
 returns varchar as
$$
import os
filename, file_ext = os.path.splitext(file_path)
return file_ext;
$$ language plpythonu;
```

PL/Python - Python modules

```
pystok=# select get_file_ext('/path/to/my/file/pystok.html');
get_file_ext
-------------
 .html
(1 wiersz)
```

PL/Python - DB Access

```
create function my_func() returns setof profile
as $$
rv = plpy.execute("SELECT * FROM profile", 5);
for x in rv:
    yield x
$$ language plpythonu;
```

PL/Python - DB Access

```
pystok=# select id, username from my_func();
 id |   username
----+--------------
  1 | Marik1234
  2 | Jarek
  3 | Andrzej
  4 | Beata
  5 | Antoni
(5 wierszy)
```

PL/Python - DB Access

```
DO
$$
  plan = plpy.prepare("INSERT INTO mytable (myval) values
($1)", ['text'])
  plpy.execute(plan, ['my value'])
$$ language plpythonu;
```

PL/Python - DB Access

- when you prepare a plan using the PL/Python module it is automatically saved
- you can use SD or GD dictionaries

```
if "plan" in SD:
   plan = SD["plan"]
else:
   plan = plpy.prepare("SELECT 1")
   SD["plan"] = plan
```

PL/Python - example

```
create table product(
 id serial primary key not null,
 product_name varchar(256),
 price decimal(7,2),
 quantity int,
);
create table sold_out(
 id serial primary key not null,
 product_id int references product (id),
 created_at timestamp default now()
);
```

```
create table error_log (
 id serial primary key not null,
 error_type varchar(64),
 error_msg varchar(512),
 raised_at timestamp default now()
)

insert into product (product_name,
price, quantity) values ('Harnas',
1.89, 10);
```

PL/Python - example

```
create or replace function check_quantity()
 returns trigger
as $$
  import smtplib

  if TD["new"]["quantity"] == 0:
       plan = plpy.prepare("INSERT INTO sold_out (product_id) VALUES ($1)",
['int'])
       plpy.execute(plan, [TD["new"]["id"]])
……
```

PL/Python - example

```
sender, receiver = ('from@gmail.com', 'to@gmail.com')
msg = "\r\n".join([
   "From: {}".format(sender),
   "To: {}".format(receiver),
   "Subject: {} sold out!".format(TD["new"]["product_name"]),
   "",
   "{}: {} sold out".format(TD["new"]["id"],
                            TD["new"]["product_name"])
])
```

PL/Python - example

```
try:
   server = smtplib.SMTP('localhost')
   server.sendmail(sender, receiver, msg)
except (smtplib.SMTPException, smtplib.SMTPAuthenticationError) as e:
  plan = plpy.prepare("INSERT INTO error_log (error_type, error_msg)
values ($1, $2)", ['text', 'text'])

  plpy.execute(plan, ('SMPTException', e))


$$ language plpythonu;
```

PL/Python - example

```
create trigger product_update
before update on product
for each row execute procedure
check_quantity();

update product set quantity=0 where id=1;
```

PL/Python - summary

- Python is a nicer language than PL/pgSQL
- ability to use Python libraries
- can communicate with other services

Questions?

piotrpalkiewicz@gmail.com