

PGDay 2009

Connettività a PostgreSQL da Python e Ruby

Gianluca Riccardi – gianluca@moonwatcher.it

4 Dicembre 2009, Pisa



Introduzione

Il talk vuole dare una panoramica di utilizzo di PostgreSQL dal linguaggio Python attraverso le specifiche di programmazione definite nella DB API 2.0 (PEP 249), mediante l'uso del DB driver psycopg2, e dal linguaggio Ruby attraverso l'utilizzo della gem ufficiale ruby-pg versione 0.8.0

Sono richieste:

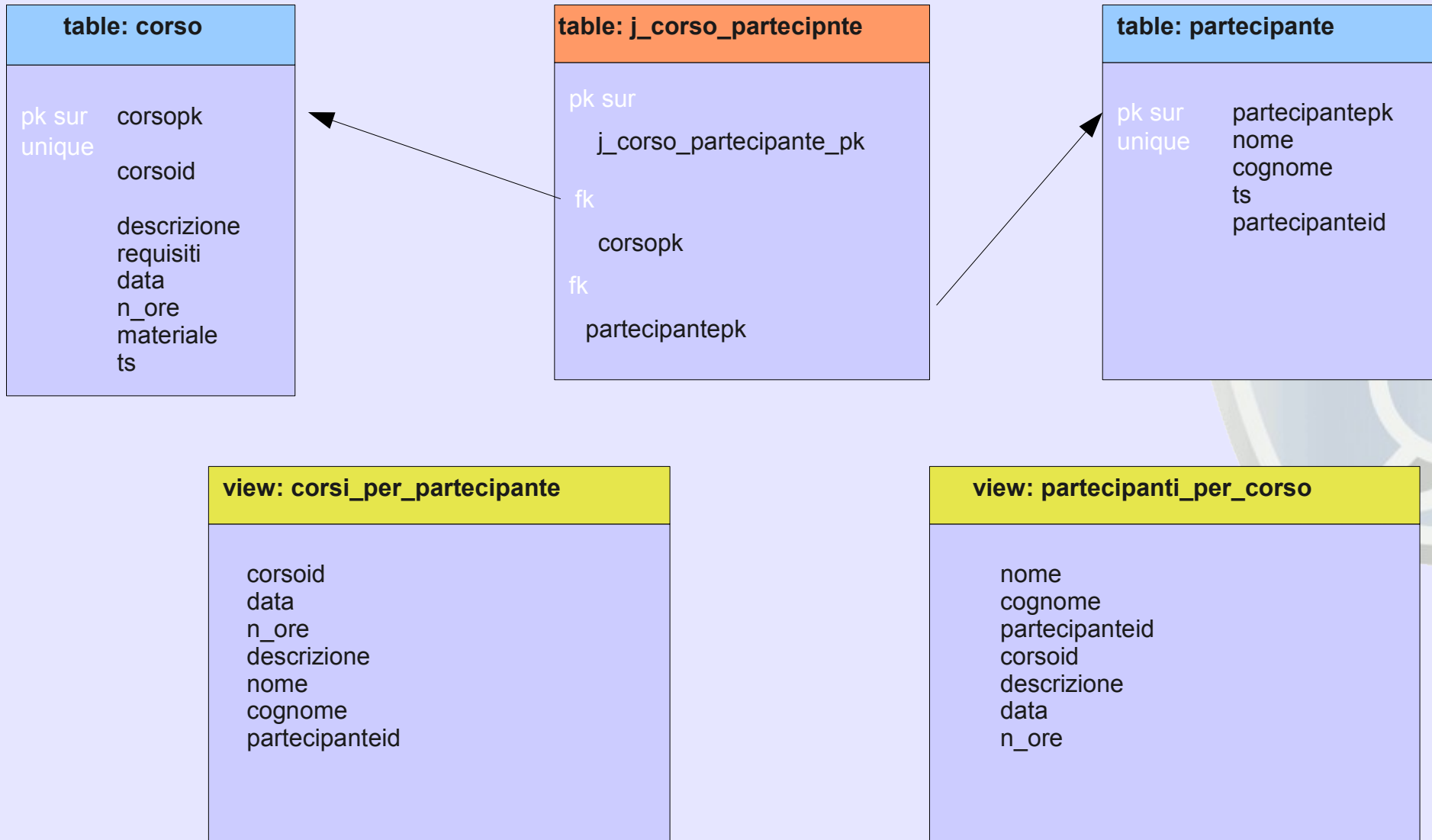
- conoscenze minime dei linguaggi in uso nelle strutture di flusso e tipi di dati come le liste e dizionari per Python e gli hashes e arrays per Ruby, per scrivere piccoli script di connessione e usare le rispettive prompt interattive;
- conoscenza minima di PostgreSQL, cosa è un DB, come crearne e come usare il terminale 'psql' su un DB già esistente per eseguire semplici statement SQL

livello: beginners



Il DB di esempio

Il DB di esempio 'pgdaydb'



Agenda

i punti focali

- **Python:**
- *Python DB API 2.0 (PEP 249)*
- *Psycopg2, Python DB adapter*
- *Python DB API 2.0 programming examples*
- **Ruby:**
- *Installazione di ruby-pg 0.8.0*
- *Requisiti e casi comuni*
- *Uso in un programma ruby*



Python DB API 2.0 (PEP 249)

- Cosa sono le Python PEPs?

un sistema attraverso cui definire guidelines, practices ed altro attorno ad un tema specifico: <http://python.org/dev/peps/>

- Cosa é la PEP-0249 ?

<http://python.org/dev/peps/pep-0249/>

“ This API has been defined to encourage similarity between the Python modules that are used to access databases. ”

in particolare:

- * **Module Interface**
- * **Connection Objects**
- * **Cursor Objects**
- * Type Objects and Constructors
- * Implementation Hints for Module Authors
- * ...



Python DB API 2.0 (PEP 249)

alcuni attributi del modulo

`apilevel` indica la DB API in uso, in questo caso vale '2.0'

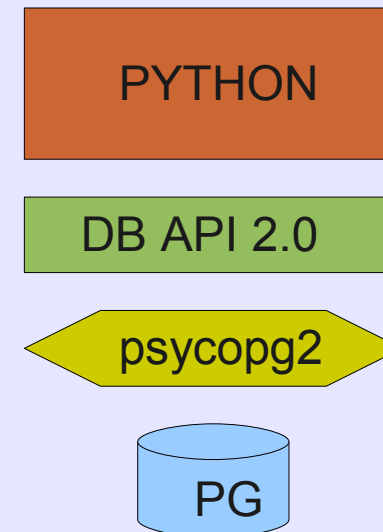
`threadsafety` 0: modulo non condivisibile
 1: modulo condivisibile fra threads
 2: connessioni sono condivisibili
 3: i cursor sono condivisibili

`paramstyle` come i parametri vengono specificati nelle queries (default:
 'pythonformat')



Psycopg2, Python DB adapter

- psycopg2 é un driver di basso livello scritto in C da Federico Di Gregorio
<http://initd.org/svn/psycopg>
- principali caratteristiche:
 - leggero, veloce e stabile come una roccia ;-)
 - realizzato per applicazioni multi-threaded
 - consente un largo uso di INSERTs e UPDATEs concorrenti
 - operazioni asincrone
 - fornisce estensioni alla DB API 2.0 (psycopg2.extensions)



Python DB API 2.0 programming

Connessione e Cursore

- importare il modulo:

```
import psycopg2
```

- definire i parametri di connessione:

```
dsn = "dbname=pgdaydb user=pgday_user password=pgday2009  
host=127.0.0.1"
```

- aprire una connessione a DB:

```
conn = psycopg2.connect(dsn)
```

- ottenere un cursore:

```
cur1 = conn.cursor()
```



Python DB API 2.0 programming

Esecuzione di queries: lettura

si usa l'oggetto cursore

- esecuzione:

```
curs1.execute("SELECT * FROM partecipante")
```

- fetching dei risultati:

```
rows = curs1.fetchall()
```

rows

```
[(1, 'Luca', 'Ferrari', datetime.datetime(2009, 9, 8, 11, 42, 38, 82147, tzinfo=<psycopg2.tz.FixedOffsetTimezone object at 0x8b9138c>), 'FRRLCUZZZZZZ'), (2, 'Diego', 'Cinelli', datetime.datetime(2009, 9, 8, 11, 43, 30, 228820, tzinfo=<psycopg2.tz.FixedOffsetTimezone object at 0x8b9140c>), None), (3, 'Emanuele', 'Zamprogno', datetime.datetime(2009, 9, 8, 11, 43, 48, 101389, tzinfo=<psycopg2.tz.FixedOffsetTimezone object at 0x8b9146c>), None), (4, 'Gianni', 'Ciolli', datetime.datetime(2009, 9, 8, 11, 43, 53, 985769, tzinfo=<psycopg2.tz.FixedOffsetTimezoneobject at 0x8b914ec>), None), (5, 'Gianluca', 'Riccardi', datetime.datetime(2009, 9, 8, 11, 44, 48, 776496, tzinfo=<psycopg2.tz.FixedOffsetTimezone object at 0x8b9156c>), None)]
```



Python DB API 2.0 programming

Esecuzione di queries: scrittura

- creazione e popolazione di una tabella:

```
books_table = "CREATE TABLE books (  
    isbn13 varchar(20) UNIQUE,  
    titolo varchar(80),  
    autori varchar(100), editrice varchar(60))"  
  
curs1.execute(books_table)  
  
curs1.execute("INSERT INTO books_table VALUES  
('9780596009250', 'Programming Python', 'M. Lutz', 'OReilly'))  
  
curs1.execute("INSERT INTO books_table VALUES  
('978-0596516178', 'The Ruby Programming Language', 'David  
Flanagan - Yukihiro Matsumoto', 'OReilly'))
```



Python DB API 2.0 programming

Esecuzione di queries: scrittura

- si deve istruire un commit perché le nostre scritture siano compiute

```
conn.commit()
```

- si usa il metodo commit() ...dell'oggetto connessione!



Python DB API 2.0 programming

Esecuzione di queries: lettura, DictCursor

- DictCursor mantiene le indicizzazioni per nome colonna

```
import psycopg2.extras
```

```
try:
```

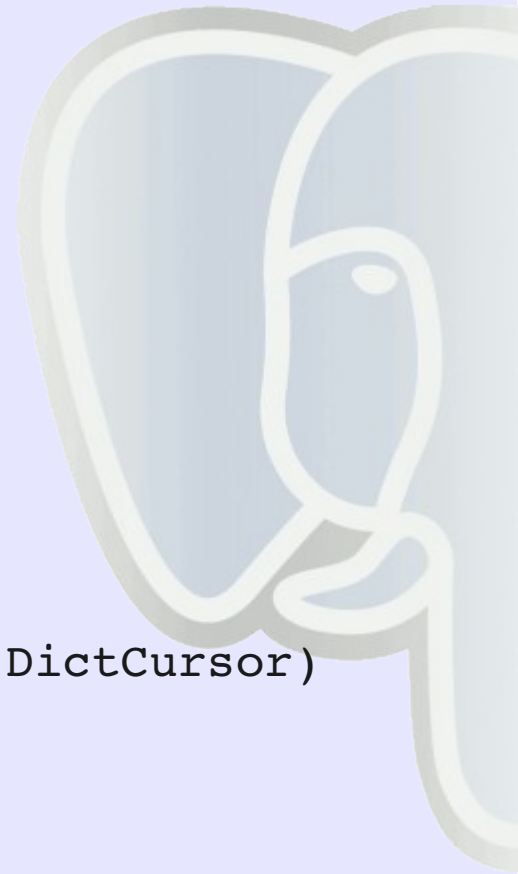
```
    conn = psycopg2.connect(dsn)
```

```
except:
```

```
    except StandardError, err:
```

```
        print "Impossibile connettersi al DB:\n", err
```

```
curs2 = conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
```



Python DB API 2.0 programming

Esecuzione di queries: lettura, DictCursor

- accesso standard:

```
 curs1.execute("select nome, cognome from  
vw_corsi_per_partecipante")
```

- accesso esteso, DictCursor

```
 curs2.execute("select nome, cognome from  
vw_corsi_per_partecipante")
```

- data retrieving:

```
 rows = curs1.fetchall()  
 rows_descr = curs2.fetchall()
```



Python DB API 2.0 programming

Esecuzione di queries: lettura, DictCursor

- contenuti:

`rows`

```
[('Diego', 'Cinelli'), ('Emanuele', 'Zamprogno'),  
 ('Gianluca', 'Riccardi'), ('Gianni', 'Ciolli'), ('Luca',  
 'Ferrari'), ('Luca', 'Ferrari')]
```

si ottiene una lista di tuple

`rows_descr`

```
[['Diego', 'Cinelli'], ['Emanuele', 'Zamprogno'],  
 ['Gianluca', 'Riccardi'], ['Gianni', 'Ciolli'], ['Luca',  
 'Ferrari'], ['Luca', 'Ferrari']]
```

si ottiene una lista di liste



Python DB API 2.0 programming

Esecuzione di queries: lettura, DictCursor

- accesso standard:

```
rows[0][0]
```

```
'Diego'
```

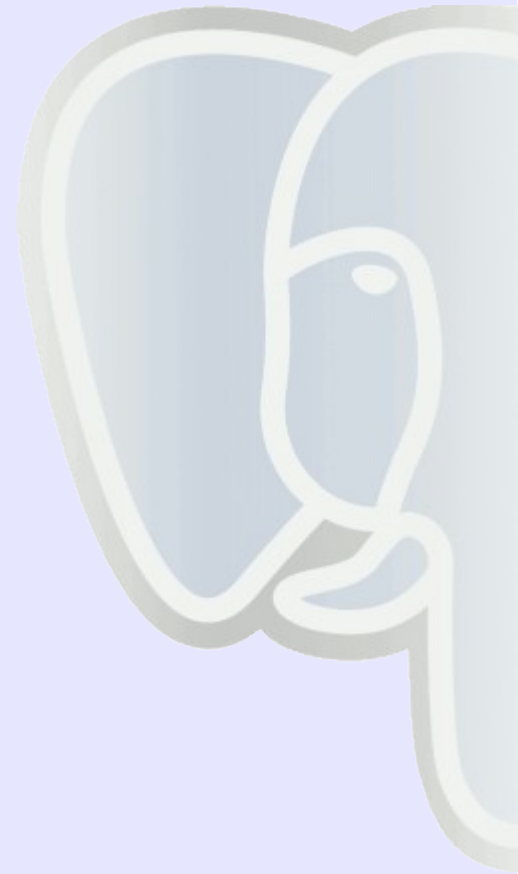
- accesso DictCursor attraverso il nome colonna:

```
rows_descr[0]['nome']
```

```
'Diego'
```

```
rows_descr[0][0]
```

```
'Diego'
```



Python DB API 2.0 programming

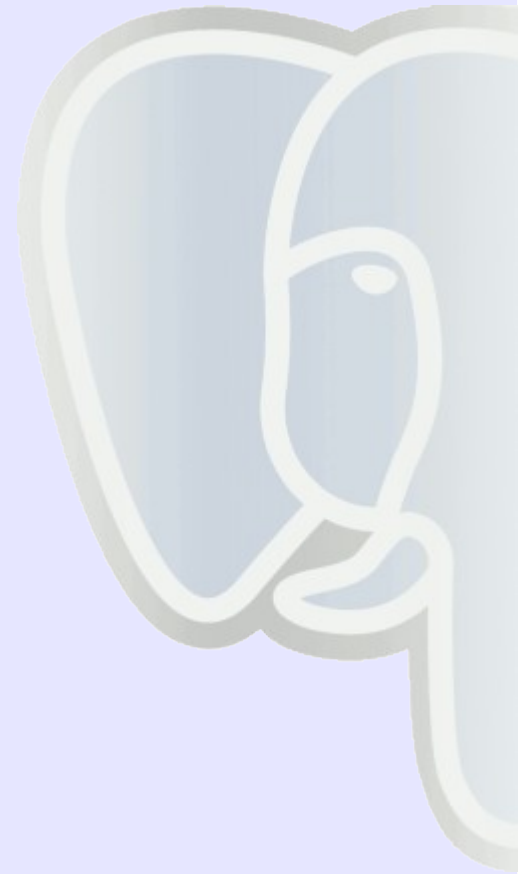
differenze

```
cursor1  
<cursor object at 0x8d5502c; closed: 0>
```

```
type(cursor1)  
<type 'psycopg2._psycopg.cursor'>
```

```
cursor2  
<cursor object at 0x8c59554; closed: 0>
```

```
type(cursor2)  
<class 'psycopg2.extras.DictCursor'>
```



Python DB API 2.0 programming

Esecuzione di queries: parametri

- si passa un secondo parametro al metodo `.execute()` del cursor

```
d = ({'nome' : 'Gabriele', 'cognome' : 'Bartolini'},  
     {'nome' : 'Simone', 'cognome' : 'Martelli'},  
     {'nome' : 'Maurizio', 'cognome' : 'Totti' } )
```

```
curs1.execute("INSERT INTO partecipante(nome, cognome) VALUES (%  
              (nome)s, %(cognome)s)", d)
```

```
conn.commit()
```



Ruby & PostgreSQL

la gem ufficiale

ruby-pg is now the official rubyforge project for the "postgres" ruby gem. See the project here:

<http://www.rubyforge.org/projects/ruby-pg>

or install the gem directly:

```
# gem install --remote postgres
```

The previous project has gone unmaintained for a long time, which lead to the fork.

This gem includes some important fixes, most notably the ability to compile against PostgreSQL 8.3.

The gem contains two modules:

- * 'postgres' -- require this module as before, you can use it without making any changes to your application. This is essentially just a fork from version 0.7.1.2006.04.06, but contains some important fixes, including the ability to build against 8.3.

- * 'pg' -- a new interface, designed to offer every feature available in libpq to Ruby, with a better API. This module is simpler, cleaner, and [...].

Regards,

Jeff Davis

13 Dicembre 2007



Ruby & PostgreSQL

installazione della gem

Dopo essere installata verrà compilata automaticamente

Pre-requisiti:

- header files:
 - PostgreSQL
 - Ruby

Installazione:

```
# gem install pg -r
```



Ruby & PostgreSQL

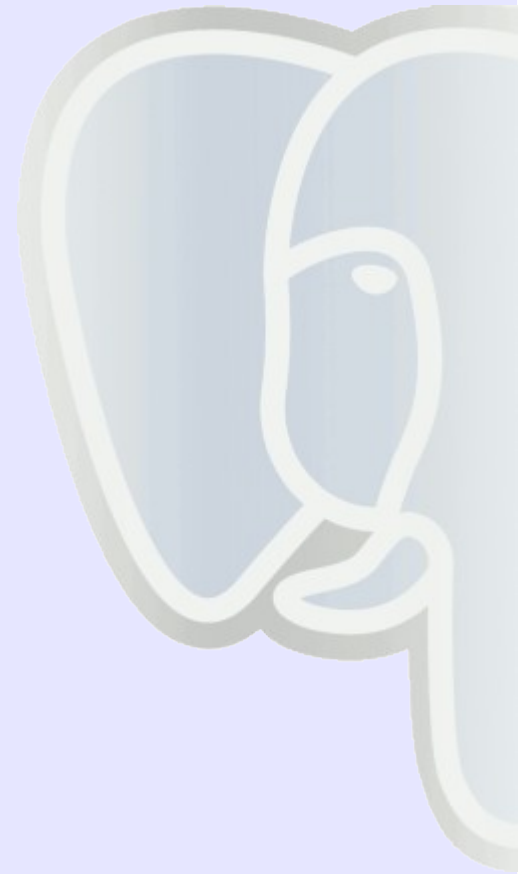
utilizzo ed inclusione in un programma

per poter usare le gem installate in un programma ruby:

```
require 'rubygems'
```

per usare la gem pg:

```
require 'pg'
```



Ruby & PostgreSQL

connessione e prima esecuzione

Data Source:

```
dsn = "dbname=pgdaydb user=pgday_user password=pgday2009  
host=127.0.0.1"
```

Connessione

```
begin  
  conn = Pgconn.connect(dsn)  
rescue PGError => err  
  puts "Errore di connessione:", err  
end  
#<Pgconn:0xb7b44bd4>  
  
results = conn.exec("select * from partecipante")  
#<PGresult:0xb7b41560>
```



Ruby & PostgreSQL

l'oggetto PGResult

results.methods

```
=> ["find", "inspect", "tap", "result_error_message", "clone", "take",  
"taguri", "getisnull", "reject", "public_methods", "__send__",  
"object_id", "minmax", "instance_variable_defined?", "num_fields",  
"equal?", "freeze", "member?", "taguri=", "each", "cmd_tuples",  
"sort", "partition", "each_cons", "extend", "any?", "each_with_index",  
"fformat", "send", "methods", "detect", "result_error_field", "hash",  
"take_while", "to_yaml_style", "getlength", "dup",  
"instance_variables", "to_enum", "clear", "collect", "fname",  
"min_by", "eql?", "cmdtuples", "sort_by", "enum_cons", "group_by",  
"id", "instance_eval", "fmod", "one?", "enum_with_index",  
"singleton_methods", "ftype", "find_index", "ntuples", "taint",  
"drop", "nparams", "instance_variable_get", "frozen?", "enum_for",  
"map", "display", "instance_of?", "max_by", "fnumber",  
"method", "grep", "to_a", "to_yaml", "oid_value", "first",  
"result_status", "instance_exec", "type", "none?", "reverse_each",  
"to_yaml_properties", "fsize", "protected_methods", "find_all", "=",  
"min", "num_tuples", "===", "drop_while", "paramtype",  
"instance_variable_set", "each_slice", "inject", "respond_to?",  
"kind_of?", "minmax_by", "ftable", "to_s", "fields", "count",  
"res_status", "class", "getvalue", "zip", "private_methods", "=~",  
"tainted?", "__id__", "select", "max", "nfields", "untaint", "nil?",  
"cmd_status", "entries", "cycle", "enum_slice", "reduce", "include?",  
"is_a?", "[]", "all?", "ftablecol"]
```



Ruby & PostgreSQL

query in lettura

Leggere i dati ottenuti attraverso l'invocazione di `conn.exec()`:

per singola riga e posizione

```
results.getvalue(0,1)
"Luca"
```

iterare tutti i risultati ottenuti

```
results.find_all.each {|x| puts x['nome']}
Luca
Diego
Emanuale
Gianni
Gianluca
```

quante righe ottenute?

```
results.num_tuples
5
```

quali sono i nomi dei campi?

```
results.fields
["partecipantepk", "nome", "cognome", "ts", "partecipanteid"]
```



Ruby & PostgreSQL

inserimento e cancellazione di dati in tabelle

Inserimento di dati in tabelle:

```
ins_res = conn.exec("insert into partecipante (nome,  
cognome) values ('Dario', 'Vignali')")
```

```
#<PGresult:0xb7ae06d4>
```

cancellazione:

```
del_res = conn.exec("delete from partecipante where  
nome='Dario'")
```

```
#<PGresult:0xb7ad9a8c>
```



Conclusioni

- cosa abbiamo visto visto:

Python

- cosa è la DB API 2.0
- come usarla
- cosa è psycopg2 come usarlo
- utilizzo base di python con PostgreSQL per queries non complesse ed esempi comuni

Ruby

- la gem ruby-pg 'pg-0.8.0'
- installazione e utilizzo base in semplici programmi
- introduzione alla struttura degli oggetti



Approfondimenti

ORMs, Object relational mappers:

- un layer fra l'applicazione ed il DB
- facilitano l'uso del db dell'SQL astrendolo
- ci 'allontanano' dal DB ...per alcuni versi
- aumentano la sicurezza delle nostre applicazioni
- sono interessanti



Approfondimenti

ORMs, Object relational mappers:

Python

- <http://www.sqlalchemy.org>
forse il migliore e più diffuso
- <http://www.sqlobject.org>
più semplice ma non meno efficiente

Ruby

- <http://ar.rubyonrails.org>
molto astratto :-) molto usato dato che è l'ORM di Ror
- <http://datamapper.org>
generalizzato, efficace, indipendente

...buona lettura e buon divertimento!



Licenza

Questa presentazione é distribuita sotto licenza Creative Commons.

<http://creativecommons.org/licenses/by-sa/2.5/it/>

