Do you talk PostGreSQL? *(en_EN)* [1)]

Ma tu parli PostGreSQL? *(it_IT)* [2)]

Ma ti pàrlito PostGreSQL? (vec_IT) [3)]

1) https://en.wikipedia.org/wiki/Main_Page - ***2)*** https://it.wikipedia.org/wiki/Pagina_principale

3) https://vec.wikipedia.org/wiki/Pajina_prinsipałe

Presentazioni

Smart Solutions

Denis Gasparin

Senior DBA and Web Developer

- Sviluppo di soluzioni software basate su PostgreSQL
- Analista e Database Administrator
- Corsi di formazione su PostgreSQL e PHP
- Contributor del driver PDO PostgreSQL per PHP
- Socio e segretario di IT-PUG

PostGreSQL è... semplice e completo!

- Rende disponibili moltissime tipi di dato e funzionalità:
  - Tipi:
    - Array
    - XML
    - JSON
    - Tipi compositi
    - Range
    - Tipi Geometrici
    - PostGIS
  - Funzionalità:
    - COPY TO/FROM
    - LISTEN/NOTIFY
    - Two Phase Commit

Ma riusciamo davvero a parlarci?

- Come sviluppatori normalmente parliamo un altro linguaggio:
  - Python
  - Java
  - PHP
  - C#
  - Ruby
  - C
  - Perl 6!
  - Swift
  - Javascript
  - Go
- Ogni linguaggio usa driver o librerie specifiche per collegarsi a PgSQL
- I dati vengono trattati di norma come tipo scalare (stringa, intero, etc)
- Non sempre tutte le features di PostGreSQL sono supportate

Un esempio

```
# estrarre l'id ed i dettagli delle persone che hanno il campo
# indirizzo

$ psql -c “SELECT id, details FROM person WHERE details ? 'address'”
```

```
<?php
$pdo = new PDO('...');
$pdo->prepare('SELECT id, details FROM person
               WHERE details ? :field_name');
$pdo->execute([':field_name' => 'address']);

// Secondo voi funziona?
```

```
// Fallisce con l'errore...

Warning: PDO::prepare(): SQLSTATE[HY093]: Invalid parameter number:
mixed named and positional parameters in php shell code on line 1
```

Un esempio - una soluzione...

```php
<?php

$pdo = new PDO('pgsql:...');
$pdo->prepare('SELECT id, details FROM person
               WHERE jsonb_exists(details, :field_name)';

$pdo->execute([':field_name' => 'address']);
```

- Per conoscere l'`operator code` associato all'operatore "?" dobbiamo:

```
SELECT oprname, oprcode FROM pg_operator WHERE oprname = '?';

 oprname |   oprcode
---------+--------------
 ?       | jsonb_exists
(1 row)
```

Un altro esempio

```
# copiare un file csv in una tabella

$ psql -c “\COPY table FROM ~/data/table.csv”
```

```
string line;

var writer = Conn.BeginTextImport("COPY table FROM STDIN");
System.IO.StreamReader file =  new System.IO.StreamReader("~/data/table.csv");

while((line = file.ReadLine()) != null)
{
    writer.Write(line);
}

writer.Close();
file.Close();
```

Qual'è il linguaggio più PostGreSQL friendly?

- Date alcune caratteristiche:
  - Stato connessione
  - Tipi JSON, Range
  - Operatore IN
  - COPY FROM/TO
  - Prepared statements
  - Tipo driver (nativo, linguaggio)
  - Performance
- Selezionati alcuni linguaggi:
  - Python
  - PHP
  - Java

Obiettivi

- Supporto delle caratteristiche
- Utile strumento per scegliere il linguaggio da utilizzare
- No flame!!

Python - drivers

- Ci sono molti driver per PgSQL:
  - Psycopg2:
    - il più popolare, nativo, Python 2.5 - 3.4
  - Pg8000:
    - Usato dal framework Web2Py, Python 2.5 - 3.0
  - Altri non più mantenuti
    - Py-postgresql
    - PyGreSQL
    - Ocpgdb, bpgsql

Fonte: https://wiki.postgresql.org/wiki/Python

Python - PsycoPG2 - Connessione

```
import psycopg2

# Connessione al database
conn = psycopg2.connect("dbname=test user=postgres")

cur = conn.cursor()

# Se autocommit è False (default), alla prima query viene inviato BEGIN
# conn.autocommit = True/False
cur.execute("SELECT version()")

if conn.status == psycopg2.extensions.STATUS_READY:
    print "No transaction, IDLE" # se autocommit = True

elif conn.status == psycopg2.extensions.STATUS_BEGIN:
    print "In Transaction: " # se autocommit = False
    print conn.get_transaction_status()
    # una di psycopg2.extensions.TRANSACTION_STATUS_* constants

else: # STATUS_PREPARED
    print "Connection ready for two phase commit"
```

Python - PsycoPG2 - JSON

```
import psycopg2.extras
from psycopg2.extras import Json

# ... Connessione al database ...
person = {'name': 'Denis Gasparin', 'age': 40}

with conn:
    with conn.cursor(cursor_factory=psycopg2.extras.DictCursor) as cur:
        rows = cur.execute("INSERT INTO person (details) VALUES (%s)", [
            Json(person)
        ])

        cur.execute("SELECT id, details::json FROM person WHERE details ? %s",
                    ['age'])
        row = cur.fetchone();

        print("Name: {0:s}".format(row["details"]["name"]));
        print("Age: {0:d}".format(row["details"]["age"]));

# Output:
# Name: Denis Gasparin
# Age: 40
```

Python - PsycoPG2 - Range

```
from datetime import datetime
from psycopg2.extras import DateTimeRange
# ... Connessione al database ...

reservation = {
    'talk': 'Do you talk PostGreSQL',
    'when': DateTimeRange(
        datetime(2015,10,23,10,45,00),
        datetime(2015,10,23,11,30,00),
        '[)'
    )
}

with conn:
    with conn.cursor() as cur:
        rows = cur.execute("INSERT INTO conference_room (talk, rs_ts)
                            VALUES (%(talk)s, %(when)s)", reservation)

        cur.execute("SELECT * FROM conference_room WHERE
                     rs_ts @> current_timestamp::timestamp")
        row = cur.fetchone();

print row
```

Python - PsycoPG2 - IN

```
# ...Connessione al database...

ids = (1,2)

with conn:
    with conn.cursor() as cur:

        cur.execute("SELECT * FROM person WHERE id IN %s", [ids])
        row = cur.fetchone();

        # Errore, non accetta NULL
        cur.execute("SELECT * FROM person WHERE id IN %s", [None])
        row = cur.fetchone();

        # SQL to the rescue!
        cur.execute("SELECT * FROM person WHERE id = ANY(%s)", [None])
        row = cur.fetchone();
```

Python - PsycoPG2 - COPY TO/FROM

```
import psycopg2

# Connessione al database
conn = psycopg2.connect("dbname=test user=postgres")

with conn:
    with conn.cursor() as cur:

        # COPY FROM
        in_file = open("~/data/person_data.csv","r")
        cur.copy_from(in_file, "person", columns=('id','details'))
        in_file.close()

        # COPY TO
        out_file = open("/tmp/person_dump.csv","w")
        cur.copy_to(out_file, "(SELECT id, details FROM person)")
        out_file.close()
```

Python - PsycoPG2 - Prepared Statements

- PsycoPG2 non li supporta al momento
- La mancata implementazione è dovuta alla mancanza di definizione dell'interfaccia nella DB API di Python
- Daniele Varrazzo ha proposto un'implementazione (che non si trova nel repo ufficiale) che può essere utilizzata ove necessario
  http://initd.org/psycopg/articles/2012/10/01/prepared-statements-psycopg/

Riepilogo

	Driver	Stato connessione	JSON	Range	IN	COPY	Prepared statements	
Python	PsycoPG2							**5/6**
Java								
PHP								

Java - drivers

- Due driver:
  - JDBC ufficiale disponibile su http://jdbc.postgresql.org
    - Il tipo di driver, secondo lo standard JDBC, è di tipo 4 (cioè scritto interamente in JAVA)
    - Supporta java 1.6, 1.7 ed 1.8
  - PGJDBC-NG disponibile su http://impossibl.github.io/pgjdbc-ng/
    - è scritto interamente in Java
    - Supporta java >= 1.7
    - Tipi di dato nativi PostGreSQL come ad esempio il Range

Java - Connessione 1/2

```
// javac -cp .:postgresql-9.4-1204.jdbc42.jar $1.java
// java -cp .:postgresql-9.4-1204.jdbc42.jar $1

import java.sql.*;
import org.postgresql.jdbc2.AbstractJdbc2Connection;
import org.postgresql.core.ProtocolConnection;

public class PgSqlConnection {

    public static void main(String[] args) throws SQLException {
        Connection conn = null;
        Integer transactionState = -1;

        String dbURL = "jdbc:postgresql://localhost/test";
        String user = "postgres";

        conn = DriverManager.getConnection(dbURL, user, null);

        ...
```

Java - Connessione 2/2 - Stato Transazione

```
    // Di default JDBC prevede autocommit abilitato

    conn.setAutoCommit(false);
    Statement st = conn.createStatement();
    ResultSet rs = st.executeQuery("SELECT version()");

    transactionState = ((AbstractJdbc2Connection) conn).getTransactionState();

    switch(transactionState) {
        case ProtocolConnection.TRANSACTION_IDLE:
            System.out.println("No transaction, IDLE");
            break;

        case ProtocolConnection.TRANSACTION_OPEN:
            System.out.println("In transaction, IDLE");
            break;

        case ProtocolConnection.TRANSACTION_FAILED:
            System.out.println("In transaction, ERROR");
            break;
    }
}
```

Java - JSON

```
import org.json.*;
import org.postgresql.util.PGobject;
...

Integer rows = 0;
JSONObject json = new JSONObject();
json.put("name", "Denis Gasparin");
json.put("age", 40);

PGobject jsonObject = new Pgobject(); // https://github.com/pgjdbc/pgjdbc/issues/265
jsonObject.setType("jsonb");
jsonObject.setValue(json.toString());

PreparedStatement ps = conn.prepareStatement("INSERT INTO person (details)
                                                VALUES (?)");
ps.setObject(1, jsonObject);
rows = ps.executeUpdate();

ps = conn.prepareStatement("SELECT * FROM person WHERE jsonb_exists(details, ?)");
ps.setString(1, "age");
ResultSet rs = ps.executeQuery(); rs.next();

json = new JSONObject(rs.getString(2));
System.out.println("Name: " + json.getString("name"));
System.out.println("Age: " + json.getLong("age"));
```

Java - JSON

```
import org.json.*;
import org.postgresql.util.PGobject;
...

Integer rows = 0;
JSONObject json = new JSONObject();
json.put("name", "Denis Gasparin");
json.put("age", 40);

PGobject jsonObject = new Pgobject();
jsonObject.setType("jsonb");
jsonObject.setValue(json.toString());

PreparedStatement ps = conn.prepareStatement("INSERT INTO person (details)
                                                VALUES (?)");
ps.setObject(1, jsonObject);
rows = ps.executeUpdate();

ps = conn.prepareStatement("SELECT * FROM person WHERE jsonb_exists(details, ?)");
ps.setString(1, "age");
ResultSet rs = ps.executeQuery(); rs.next();

json = new JSONObject(rs.getString(2));
System.out.println("Name: " + json.getString("name"));
System.out.println("Age: " + json.getLong("age"));
```

Java - Range

```
import java.util.Calendar;

Calendar start = Calendar.getInstance();
start.set(2015, 9, 23, 10, 45, 0); // I mesi in java sono 0-based

Calendar end = Calendar.getInstance();
end.set(2015, 9, 23, 11, 30, 0);

PreparedStatement ps = conn.prepareStatement("INSERT INTO conference_room
                                              (talk, rs_ts) VALUES
                                              (?, tsrange(?, ?, ?))");
ps.setString(1, "Do you talk PostGreSQL");
ps.setTimestamp(2, new Timestamp(start.getTimeInMillis()));
ps.setTimestamp(3, new Timestamp(end.getTimeInMillis()));
ps.setString(4, "[)");
rows = ps.executeUpdate();

ps = conn.prepareStatement("SELECT * FROM conference_room WHERE
                            rs_ts @> current_timestamp::timestamp");
ResultSet rs = ps.executeQuery();
rs.next();
System.out.println("Reservation Range: " + rs.getString(2));
// Reservation Timestamp: ["2015-10-10 10:45:00.017","2015-10-10 23:59:00.017")
```

Java - IN

```
Integer[] ids = new Integer[]{1, 2, 3};
String sql = "SELECT * FROM person WHERE id = ANY (?::int[])";

PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setArray(1, conn.createArrayOf("int4", ids));

ResultSet rs = pstmt.executeQuery();
rs.next();

JSONObject json = new JSONObject(rs.getString(2));
System.out.println("Name: " + json.getString("name"));
System.out.println("Age: " + json.getLong("age"));
```

Java - COPY TO/FROM

```java
import java.io.*;
import org.postgresql.copy.CopyManager;
import org.postgresql.core.BaseConnection;

public class PgCopy {

    public static void main(String[] args) throws SQLException, IOException {

        // ... Connessione al db ...

        CopyManager copyManager = new CopyManager((BaseConnection) conn);

        FileReader inFile = new FileReader("~/data/person_data.csv");
        copyManager.copyIn("COPY person(id, details) FROM STDIN", inFile);
        inFile.close();

        FileWriter outFile = new FileWriter("/tmp/person_dump.csv");
        copyManager.copyOut("COPY (SELECT id, details FROM person) TO STDOUT",
                            outFile
        );
        outFile.close(); // Importante, altrimenti non scrive nulla!
    }
}
```

Java - Prepared Statements

```
import org.postgresql.PGStatement;
...

JSONArray data = new JSONArray("[" +
            "{\"name\": \"Michael Stonebraker\",\"age\": 72}," +
            "{\"name\": \"Andrew Yu\", \"age\": 49}," +
            "{\"name\": \"Jolly Chen\",\"age\": 48}]");

PGobject jsonObject = new Pgobject(); // Query a db, in cache
jsonObject.setType("jsonb");

PreparedStatement ps = conn.prepareStatement("INSERT INTO person (details)
                                              VALUES (?)");
PGStatement pgPs = (PGStatement) ps;
pgPs.setPrepareThreshold(1); // Anche a livello di connessione

for(i=0;i<data.length();i++) {
    jsonObject.setValue(data.getJSONObject(i).toString());
    ps.setObject(1, jsonObject);
    ps.executeUpdate();

    System.out.println("Inserted: " + data.getJSONObject(i).getString("name"));
}
```

Riepilogo

	Driver	Stato connessione	JSON	Range	IN	COPY	Prepared statements	
Python	PsycoPG2							**5**
Java	JDBC 4.2							**4,5**
PHP								

PHP - drivers

- Due modalità di connessione:
  - Funzioni: pg_*
    - Prime ad essere state implementate
    - No parametri posizionali/nominali
  - PDO::PgSQL
    - Interfaccia ad oggetti standard
    - Parametri
    - Usata da diversi ORM (Doctrine, Propel)
- Entrambe le modalità sono native (basate su LibPQ)

PHP - PDO - Connessione

```php
<?php

$conn = new PDO("pgsql:dbname=test;user=postgres");

$conn->query("SELECT version()");

echo $conn->inTransaction() ?
    "In transaction: IDLE\r\n" :
    "No transaction, IDLE\r\n";

$conn->beginTransaction();

echo $conn->inTransaction() ?
    "In transaction: IDLE\r\n" :
    "No transaction, IDLE\r\n";

$conn->commit();

// PHP prima di 5.4, inTransaction() non usa la relativa funzione di libPQ
```

PHP - PDO - JSON

```php
<?php

$conn = new PDO("pgsql:dbname=test;user=postgres");

$json = [
    'name' => 'Denis Gasparin',
    'age' => 40
];

$s = $conn->prepare("INSERT INTO person (details) VALUES (?)");
$rows = $s->execute([json_encode($json)]);

$s = $conn->prepare("SELECT * FROM person WHERE jsonb_exists(details, :field)");
$s->execute([':field' => 'age']);
$r = $s->fetch(PDO::FETCH_ASSOC);

$jsonResult = json_decode($r['details'], true);
echo "Name: {$jsonResult['name']}\n";
echo "Age: {$jsonResult['age']}\n";

// Name: Denis Gasparin
// Age: 40
```

PHP - PDO - Range

```php
// Attenzione: di default PDO non alza eccezione ma ritorna bool
//             nell'esecuzione delle query
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$s = $conn->prepare("INSERT INTO conference_room (talk, rs_ts)
                     VALUES (:talk, tsrange(:start, :end, :exc_inc))");

$s->execute([
    ":talk" => "Do you talk PostGreSQL",
    ":start" => (new DateTime("2015-10-23 10:45:00"))->format("Y-m-d H:i:s"),
    ":end" => (new DateTime("2015-10-23 11:30:00"))->format("Y-m-d H:i:s"),
    ":exc_inc" => "[)"
]);

$s = $conn->prepare("SELECT * FROM conference_room
                     WHERE rs_ts @> current_timestamp::timestamp");
$s->execute();
$r = $s->fetch(PDO::FETCH_ASSOC);

echo "Reservation Range: {$r["rs_ts"]}\n";

// Output:
//  Reservation Range: ["2015-10-23 10:45:00","2015-10-23 11:30:00")
```

PHP - PDO - IN

```php
<?php

$conn = new PDO("pgsql:dbname=test;user=postgres");

$params = array(1, 2, 3);

// Attenzione: è obbligatorio fare il cast del parametro ad intero
//             oppure al tipo utilizzato
$placeHolders = implode(',', array_fill(0, count($params), '?::int'));

$s = $conn->prepare("SELECT * FROM person WHERE
                     id = ANY(ARRAY[{$placeHolders}])");

$s->execute($params);
$r = $s->fetch(PDO::FETCH_ASSOC);
$jsonResult = json_decode($r['details'], true);

echo "Name: {$jsonResult['name']}\n";
echo "Age: {$jsonResult['age']}\n";
```

PHP - PDO - COPY TO/FROM

```php
<?php

$conn = new PDO("pgsql:dbname=test;user=postgres");

$conn->pgsqlCopyFromFile("person",
                         "~/data/person_data.csv",
                         NULL, // Field Separator, default is tab
                         NULL, // Null, default is \N
                         "id, details"
);

$conn->pgsqlCopyToFile("(SELECT id, details FROM person)",
                       "/tmp/person_dump.csv"
);
```

PHP - PDO - Prepared Statements

- PDO usa di default i prepared statements di PostGreSQL
- L'unico caso in cui non sono usati è nel caso del metodo PDO::exec():
  - esegue direttamente la query a db
  - no escape parametri
  - non può ritornare un recordset
- E' possibile disabilitare il comportamento con:

```
$conn->setAttribute(PDO::ATTR_EMULATE_PREPARES, 1);
```

Riepilogo

	Driver	Stato connessione	JSON	Range	IN	COPY	Prepared statements	
Python	PsycoPG2							**5**
Java	JDBC 4.2							**4,5**
PHP	PDO	PHP >= 5.4						**3,5**

Benchmarks

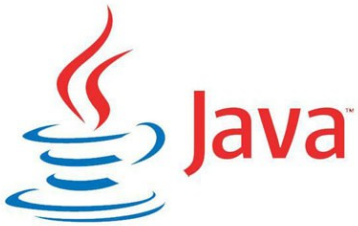
- Obbiettivo:
  - verificare l'overhead dovuto al parsing dei parametri manuale o del driver
- Test con inserimento di 100.000 righe
  - con e senza prepared statements (ove disponibili)
  - con e senza interpretazione del driver (ove disponibile)
- Risultato in percentuale per ogni singolo linguaggio

Risultati

(*) With Parameter Substitution
(**) Without Driver Parameter Substitution

	Prepared	Not Prepared
(*) With		100,00%
(**) Without		97,20%

	Prepared	Not Prepared
(*) With	91,86%	100,00%
(**) Without		99,89%



	Prepared	Not Prepared
(*) With	93,64%	100,00%
(**) Without		103,34%

PgBabylon

- Obiettivi:
  - Integrare le feature specifiche di PostgreSQL all'interno dei driver dei linguaggi di programmazione più utilizzati
  - Algoritmi di conversione da/per PostGreSQL:
    - ottimizzazione degli algoritmi
    - condivisione tra i vari linguaggi
  - Ampliare la diffusione di PostgreSQL

PHP - PgBabylon

- Estensione (User-land) delle classi PDO e PDOStatement
- Aggiunge supporto IN/OUT dei tipi di dato:
  - JSON, Array, DateTime (Timestamp, Date)
- Supporto nativo all'operatore IN
- Retro-compatibile con la libreria PDO
- Semplice da integrare nei progetti esistenti
- Disponibile via `composer`
- Sorgente su: https://github.com/rtshome/pgbabylon

PgBabylon - Esempio

```php
<?php
use PgBabylon\PDO;
use PgBabylon\DataTypes;

$pdo = new PDO("pgsql:dbname=test;user=postgres");
$s = $pdo->prepare("INSERT INTO person(details, insertion_date)
                    VALUES (:person, :ins_date) RETURNING *");

$person = [
    "name" => "Denis Gasparin",
    "age" => 40
];

$s->execute([
    ':person' => DataTypes\JSON($person),
    ':ins_date' => DataTypes\Date(new DateTime())

// setColumnTypes() is a PgBabylon extension
$s->setColumnTypes([
    'details' => PDO::PARAM_JSON,
    'insertion_date' => PDO::PARAM_DATE

$r = $s->fetch(PDO::FETCH_ASSOC);
```

Thank You!

https://github.com/rtshome/pgbabylon

Smart Solutions

denis@gasparin.net

@rtshome

http://www.gasparin.net