

# Stored procedures in PL/pgSQL

Gianni Ciolli

`gianni.ciolli@2ndquadrant.it`

PGDay 2009, Pisa



## Licenza

- Quest'opera è stata rilasciata sotto la licenza **Creative Commons Attribuzione-Non commerciale-Non opere derivate 3.0 Unported**



- Per leggere una copia della licenza visita il sito web

<http://creativecommons.org/licenses/by-nc-nd/3.0/>

o spedisce una lettera a

*Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.*



# Programma

- 1 **Introduzione**
- 2 **PL/pgSQL in particolare**
  - Caratteristiche
  - Il linguaggio
  - I comandi
- 3 **In conclusione**



## Introduzione

Questo (mini)corso è interamente dedicato al linguaggio procedurale PL/pgSQL.

Dapprima esporremo sommariamente alcune informazioni di carattere generale su PL/pgSQL e sugli altri linguaggi procedurali distribuiti con PostgreSQL.

Successivamente passeremo in rassegna le funzionalità di PL/pgSQL, con un duplice scopo:

- far capire meglio cosa si può fare con tale linguaggio,
- e suggerire gli ambiti di applicazione (non tutto ciò che è possibile è consigliato).



## L'autore

- Dottorato di ricerca in Matematica (2000-2003)
- Presidente e socio fondatore dell'Associazione PLUG - Prato Linux User Group (2001-2004)
- Socio di ITPUG
- Usa PostgreSQL dal 2006
- Consulente 2ndQuadrant Italia dal 2008



## Rapidissima introduzione al contesto

- Server programming
- User-defined functions
- Procedural languages
- Installare un linguaggio in un database



## Caratteristiche di PL/pgSQL

- linguaggio procedurale
- interpretato dall'esecutore SQL del server
- riduce il traffico client-server
- usa esattamente gli stessi tipi, operatori e funzioni disponibili per le normali query SQL
- le funzioni possono avere diversi tipi in uscita:
  - void
  - un tipo (semplice, oppure composto, e. g. una riga)
  - più righe (clausola SETOF)
- polimorfismo a una dimensione (ottenuto con i tipi `anyelement`, `anyarray`, `anynonarray`, `anyenum`)
- tipo `RECORD`
- comodità: valori `IN` e `OUT`



## Dichiarazione di una funzione PL/pgSQL

### Esempio:

```
CREATE FUNCTION f(x integer, y text)
RETURNS integer AS $$
DECLARE
  :
BEGIN
  :
END;
$$ LANGUAGE plpgsql;
```



## Struttura di una funzione PL/pgSQL, 1/2

Un *blocco* ha la seguente forma:

```
DECLARE
  dic1
  ⋮
  dicn
BEGIN
  stmt1
  ⋮
  stmtn
END;
```



## Struttura di una funzione PL/pgSQL, 2/2

Una funzione:

- ha esattamente un blocco principale
- può avere sottoblocchi (ad esempio, per definire variabili locali)



## Commenti al codice

I commenti si fanno in due modi:

- come in SQL:

```
-- Questo commento e`  
-- in stile SQL
```

- come in C:

```
/* Questo invece e` un  
commento in stile C */
```

(però in PL/pgSQL non si possono annidare i commenti!)



## Dichiarazioni di variabili, 1/2

- esempi:

```
user_id integer;  
quantity numeric(5);  
url varchar;
```

- tipi definiti implicitamente (ottima prassi)

```
myrow tablename%ROWTYPE;  
myfield tablename.columnname%TYPE;
```



## Dichiarazioni di variabili, 2/2

- con questa dichiarazione

```
row RECORD;
```

la variabile assumerà dinamicamente il tipo che le verrà assegnato

- possono esserci dei default

```
quantity integer DEFAULT 32;  
url varchar := 'http://mysite.com';  
user_id CONSTANT integer := 10;
```

i quali sono calcolati ogni volta che si entra nel blocco



## Comandi PL/pgSQL

Adesso passeremo in rassegna i comandi (statements) di PL/pgSQL.



## Comandi PL/pgSQL: assegnazioni

- A una variabile si può assegnare un valore:

```
variable := expression;
```

- Esempi:

```
tax := subtotal * 0.06;  
my_record.user_id := 20;
```



## Comandi PL/pgSQL: eseguire query prive di output, 1/2

Una query che sia *di per sé* priva di output si può eseguire semplicemente scrivendola:

```
DECLARE
    key TEXT;
    delta INTEGER;
BEGIN
    ...
    UPDATE mytab SET val = val + delta
        WHERE id = key;
```



## Comandi PL/pgSQL: eseguire query prive di output, 2/2

Se invece si vuole eseguire una query di tipo SELECT senza essere però interessati all'output (ad esempio per invocare una funzione), si deve usare il comando PERFORM invece di SELECT:

```
PERFORM create_mv (my_query,  
                  'cs_session_page_requests_mv');
```



## Comandi PL/pgSQL: assegnare valori forniti da query

- Per assegnare alle variabili dei valori forniti da una query si usa il costrutto **SELECT INTO**:

```
SELECT select_expression  
INTO target [STRICT] FROM ...;
```



## Comandi PL/pgSQL: comandi dinamici

- Il comando EXECUTE permette di eseguire comandi *dinamici*, nel senso che il testo del comando che poi verrà eseguito può essere costruito dal programma stesso
- Sintassi:

```
EXECUTE command-string  
[ INTO [STRICT] target ];
```



## Comandi PL/pgSQL: informazioni sull'esito

- Il comando seguente permette di ottenere informazioni sull'esito delle query:

```
GET DIAGNOSTICS integer_var = ROW_COUNT;
```

- La variabile FOUND è una conveniente scorciatoia:

```
IF FOUND THEN ... ELSE ... END IF;
```

questo costrutto, usato insieme a PERFORM, permette di verificare facilmente l'esistenza di record che soddisfano determinate condizioni.



## Comandi PL/pgSQL: il comando più banale

- Il comando NULL non fa niente:

```
NULL;
```



## Comandi PL/pgSQL: restituire valori

- il comando RETURN permette di restituire un valore
- variante:  
RETURN NEXT e/o RETURN QUERY, e infine RETURN.



## Comandi PL/pgSQL: controllo del flusso

- Sono supportate varie forme di IF:
- IF ... THEN ... END IF;
- IF ... THEN ... ELSE ... END IF;
- IF ... THEN ... ELSIF ... THEN ... END IF;
- etc.



## Comandi PL/pgSQL: cicli

- LOOP ... END LOOP;
- EXIT
- CONTINUE
- WHILE ... LOOP ... END LOOP;
- FOR i IN 1 .. 10 LOOP ... END LOOP;
- È disponibile infine una variante che scorre i risultati di una query (più semplice da usare rispetto ai cursori)



## Comandi PL/pgSQL: gestione degli errori, 1/2

- i blocchi hanno una parte (facoltativa) di gestione delle eccezioni:

```
DECLARE
  ⋮
BEGIN
  ⋮
EXCEPTION
  WHEN cond1 THEN ...
  WHEN cond2 THEN ...
  ⋮
END;
```



## Comandi PL/pgSQL: gestione degli errori, 2/2

- Le condizioni possono fare uso dei codici di errore di PostgreSQL
- esempi:
  - `division_by_zero`
  - `no_data`
  - `not_null_violation`
  - `QUERY_CANCELED`
  - ...
- `OTHERS` comprende tutte le eccezioni, esclusa `query_canceled` (è saggio così).



## Comandi PL/pgSQL: cursori

- PL/pgSQL ha la possibilità di definire e usare cursori
- Due applicazioni principali:
  - scorrere i risultati di una query (ma questo si può fare anche e più semplicemente con il comando FOR)
  - restituire un puntatore a dei dati



## Comandi PL/pgSQL: messaggi di varia priorità

- si possono inviare messaggi all'utente e/o al log:

`RAISE level format exp1 ... expn`

- esempi:

```
RAISE NOTICE 'attenzione: %', v_job_id;
```

```
RAISE EXCEPTION 'ID nullo %', user_id;
```



## Riferimenti

Nella documentazione ufficiale di PostgreSQL c'è un capitolo dedicato a PL/pgSQL.



Ci sono domande?



**Grazie** per la vostra attenzione!

*(Gianni Ciolli)*

