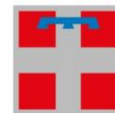




Cofinanziato  
dall'Unione europea



REGIONE  
PIEMONTE

# FinTech Software Developer

## Basi di dati SQL

Docente: Roi Davide Simone

Titolo argomento: Sviluppo su Postgresql – Scrittura di Stored Procedures e Trigger  
(1° parte)

Roi Davide  
Dispense

# PL/pgSQL

PL/pgSQL è un'estensione del linguaggio SQL utilizzato da PostgreSQL. È un linguaggio procedurale che consente di creare funzioni definite dall'utente, **Stored Procedure** (procedure/funzioni memorizzate).

PL/pgSQL è stato progettato per estendere le funzionalità del server PostgreSQL creando oggetti server con logica complessa.

Grazie alla elasticità del linguaggio di programmazione procedurale (cicli, condizioni, ecc..) e la potenza del SQL di accedere ai dati del DB (interrogazioni, inserimenti, modifiche, cancellazioni, ecc..) è possibile scrivere programmi anche molto complessi che eseguono una serie di operazioni e salvare il tutto sul server PostgreSQL.

In PostgreSQL, una stored procedure può essere creata utilizzando la sintassi **CREATE PROCEDURE** e se viene incluso un nome di schema, la procedura viene creata nello schema specificato.

Per richiamare una stored procedure all'interno di altre procedure, si utilizza la sintassi **CALL**. in postgresSQL vantaggi:

- automazione
- versatilità/disponibilità

# Stored Procedures PostgreSQL

(Esempio di creazione di una Funzione o Procedura)

solo create function non consente sovrascrittura  
quindi meglio questa

```
create [or replace] function nome_procedura (lista_parametri) [RETURNS tipo_dato_restituito]
```

```
language plpgsql
```

```
as $$      <-- inizio blocco di codice
```

```
declare
```

```
    -- dichiarazione delle variabili
```

```
begin
```

```
    -- contenuto della procedura
```

```
    RETURN valore_di_ritorno;
```

```
end; $$    <--- fine blocco di codice
```

```
create [or replace] procedure nome_procedura (lista_parametri)
```

```
language plpgsql
```

```
as $$
```

```
declare
```

```
    -- dichiarazione delle variabili
```

```
begin
```

```
    -- contenuto della procedura
```

```
end; $$
```

Fuori funzione: call nome f/p(valori)

N.B:

Una funzione prevede  
di restituire sempre un  
valore di ritorno.

Una procedura non  
prevede la gestione di  
valori di ritorno

# Stored Procedures PostgreSQL

(Esempio di creazione di una Funzione o Procedura)

Roi Davide  
Dispense

```
create or replace function raddoppia (par_1 numeric) RETURNS numeric
language plpgsql
as $$
declare
begin
    return par_1 * 2;
end; $$
```

Per testarla scrivere in una finestra SQL:

```
select
    raddoppia(12.3);
```

# Stored Procedures PostgreSQL

(Esempi vari di variabili dichiarabili in una procedura)

user\_id **integer**; --variabile di tipo numerico intero  
 quantity **numeric(5)**; --variabile di tipo numerico lungo 5 cifre  
 url **varchar(500)**; --variabile di tipo varchar(500)  
 myrow tablename **%ROWTYPE**; --variabile che può contenere una specifica riga di tabella  
 myfield tablename.columnname **%TYPE**; --variabile che può contenere uno specifico tipo dato di colonna  
 arow **RECORD**; -- variabile che può contenere un intero record di tabella      tabella generica, riga dinamica...?  
 dataora\_modifica **timestamp**; -- variabile di tipo timestamp

Le variabili possono essere anche inizializzate al momento della dichiarazione:

quantity **integer** **DEFAULT 32**;      = 32 come standard  
 url **varchar** **:= 'http://mysite.com'**;  
 transaction\_time **CONSTANT** **timestamp with time zone** **:= now()**;

Esempio tratto dalla documentazione ufficiale PostgreSQL: <https://www.postgresql.org/docs/current/plpgsql-declarations.html>

# Stored Procedures PostgreSQL

(cicli e condizioni)

Roi Davide  
Dispense

```
for var_record in
```

```
  select
```

```
    colonna1,
```

```
    fk_colonna2
```

```
  from
```

```
    tabella1
```

```
Loop
```

```
  /*se colonna1 vale più di 10 aggrino a 1 il record della tabella padre*/
```

```
  if var_record.colonna1 > 10
```

```
    update tabella2 set colonna2 = 1 where tabella2.pk_colonna1 = var_record.fk_colonna2;
```

```
  /*diversamente aggrino a 0 il record della tabella padre*/
```

```
  else
```

```
    update tabella2 set colonna2 = 0 where tabella2.pk_colonna1 = var_record.fk_colonna2;
```

```
  end if;
```

```
end loop;
```

## Fonti:

- Documentazione ufficiale di Postgresql  
<https://www.postgresql.org/docs/>

Fine della Presentazione