

Uso de funciones PL/SQL en postgresQL con C#

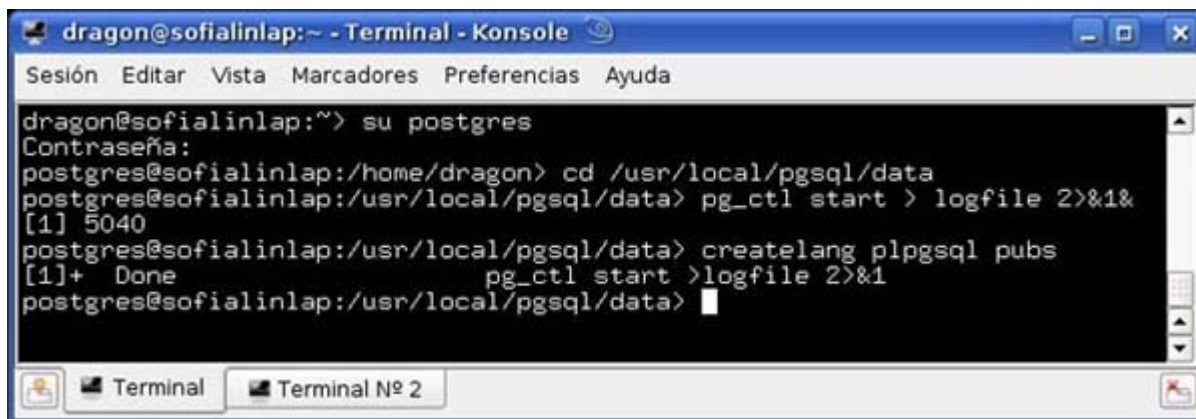
por Martín Márquez <xomalli@gmail.com>

Una función (function) en PostgreSQL son sentencias SQL agrupadas y precompiladas para ejecutarse en bloque dentro del servidor, a diferencia de las consultas SQL donde cada consulta es procesada en tiempo de ejecución por el servidor, las funciones procedurales son compilados cuando son creados, ya que el servidor asume que serán ejecutados más de una vez, un función ofrece las siguientes ventajas:

- No sobrecarga la comunicación cliente/servidor al evitar enviar una consulta tras otra, en su lugar procesa una consulta tras otra y envía únicamente el resultado.
- Cuando y se ejecuta la primera vez se crea un plan preparado de ejecución, las siguientes ejecuciones reutilizan el plan preparado.
- Agrega estructuras de control y capacidad de calculo al lenguaje SQL.
- Las mismas consultas están disponibles para varias aplicaciones.
- Seguridad los datos solo están accesibles mediante las funciones y evita el uso de SQL injection.

De los lenguajes más utilizados para crear funciones en postgresQL, se encuentra PL/pgSQL, el cual se distribuye como un módulo cargable junto con postgresQL, para emplearlo en nuestra base de datos es necesario darlo de alta, de la siguiente manera.

Fig 1 Agregando el soporte PL/SQL

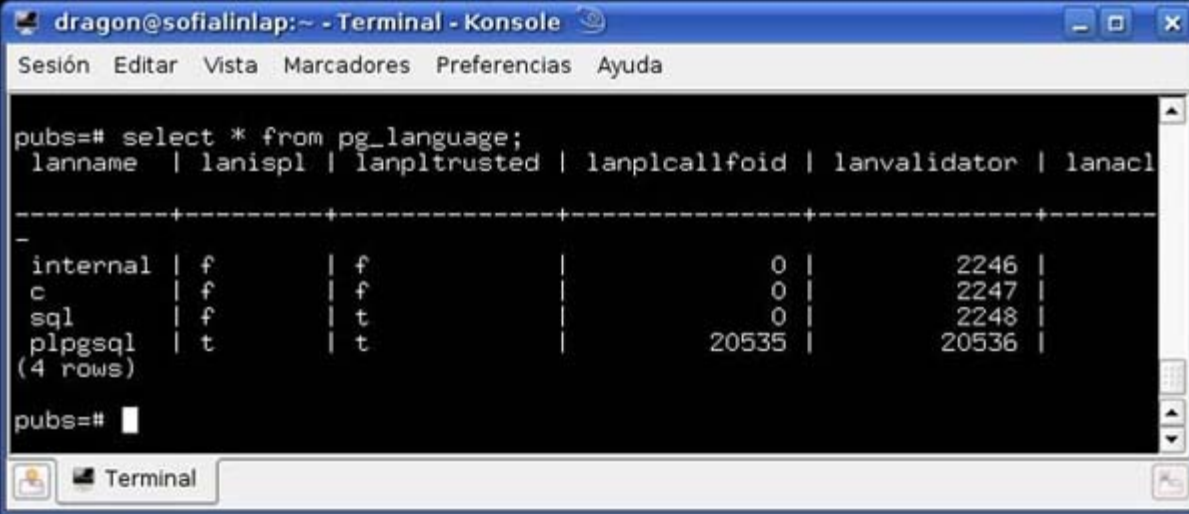


```
dragon@sofialinlap:~ - Terminal - Konsole
Sesión Editar Vista Marcadores Preferencias Ayuda

dragon@sofialinlap:~> su postgres
Contraseña:
postgres@sofialinlap:/home/dragon> cd /usr/local/pgsql/data
postgres@sofialinlap:/usr/local/pgsql/data> pg_ctl start > logfile 2>&1
[1] 5040
postgres@sofialinlap:/usr/local/pgsql/data> createlang plpgsql pubs
[1]+  Done                  pg_ctl start >logfile 2>&1
postgres@sofialinlap:/usr/local/pgsql/data> 
```

Revisamos si ya lo tenemos disponible en nuestra base de datos para utilizarlo La sintaxis de PL/pgSQL (similar al lenguaje PL/SQL de Oracle)

Fig 2 Comprobando la instalación del lenguaje PL/SQL



A terminal window titled "dragon@sofialinlap:~ - Terminal - Konsole" showing the execution of a SQL query. The query is "select * from pg_language;" and the output is a table with 6 columns: lanname, lanispl, lanpltrusted, lanplcallfoid, lanvalidator, and lanaccl. The output shows 4 rows of data for the languages internal, c, sql, and plpgsql. The plpgsql row shows a call foid of 20535 and a validator of 20536.

```
dragon@sofialinlap:~ - Terminal - Konsole
Sesión  Editar  Vista  Marcadores  Preferencias  Ayuda

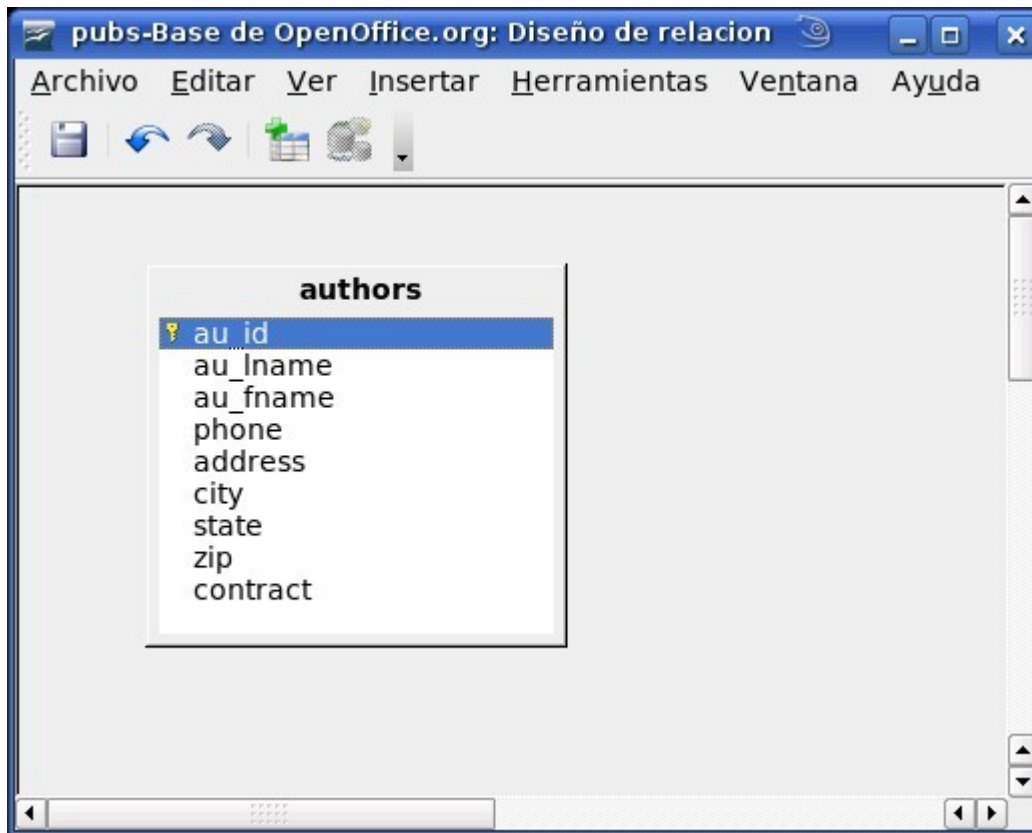
pubs=# select * from pg_language;
 lanname | lanispl | lanpltrusted | lanplcallfoid | lanvalidator | lanaccl
-----+-----+-----+-----+-----+-----
internal | f       | f           |              0 |              2246 |
c       | f       | f           |              0 |              2247 |
sql     | f       | t           |              0 |              2248 |
plpgsql | t       | t           |          20535 |          20536 |
(4 rows)

pubs=#
```

Utilizando PL/pgSQL con C#

En este ejemplo usaremos PL/pgSQL y C# para resolver un requerimiento practico como seria relacionar la columna city de nuestra tabla authors en nuestra base de datos con una tabla llamada cities donde se encontrará la información de la columna ciudad mas un identificador.

Fig 3 La tabla autores



La relación deberá de quedar de la siguiente manera, donde la columna *city* se debe cambiar por la clave primaria de la tabla *cities* que tendrá como clave primaria la clave de la ciudad y una columna adicional llamada *city* que contendrá el nombre de la ciudad.

```
CREATE TABLE cities(  
idcity varchar(5) PRIMARY KEY,  
city varchar(20) NOT NULL UNIQUE  
);
```

Ahora usamos la siguiente función *AddCities(varchar)* para tomar los valores de la columna *city* en la tabla *authors*, crear un identificador único para la llave primaria, insertar ese valor de clave primaria junto con el nombre de la ciudad y por último sustituir los valores en la columna *city* y reemplazarlos con el valor de la llave primaria en la tabla *cities*.

```
CREATE FUNCTION AddCities(varchar) RETURNS VARCHAR AS  
'DECLARE  
hay record;  
nume varchar;  
BEGIN  
SELECT INTO hay count(city) FROM cities WHERE city = $1;  
IF hay.count = 0  
THEN
```

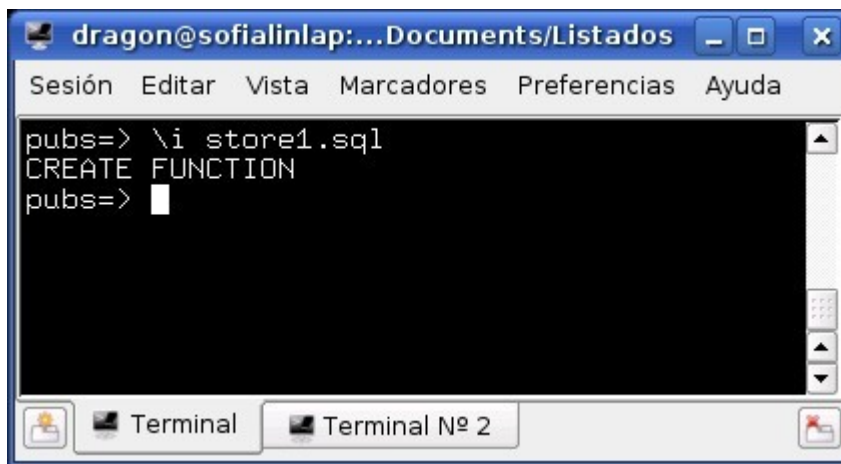
```

INSERT INTO cities(idcity,city)
VALUES(substring($1,1,3) || substring(random() * 1000,1,2),$1);
SELECT INTO nume idcity FROM cities WHERE city = $1;
UPDATE authors SET city = nume WHERE city = $1;
RETURN nume;
END IF;
IF hay.count > 0
THEN
SELECT INTO nume idcity FROM cities WHERE city = $1;
UPDATE authors SET city = nume WHERE city = $1;
RETURN nume;
END IF;
END;
' LANGUAGE 'plpgsql';

```

Guardamos nuestra función en un archivo llamado *store1.sql* y lo ejecutamos para crearla dentro del servidor.

Fig 4 Guardando la función en un archivo



Una vez creada nuestra función mostraremos como emplearla desde C# con el siguiente:

```

using System;
using Gtk;
using System.Text;
using System.Data;
using Npgsql;
using NpgsqlTypes;

namespace PgForm {
class PgForm : Window {
Label lbMsg = new Label("DEBUG: ");
Entry txtNameFunction = new Entry();
Entry txtArg = new Entry();

public PgForm() : base("Ejecutar funciones"){
BorderWidth = 8;
SetDefaultSize(208,220);
this.DeleteEvent += new DeleteEventHandler(OnWindowDelete);
Frame frame = new Frame ("Ejecutar funciones");

```

```

Add (frame);
VBox MainPanel = new VBox (false, 8);
MainPanel.BorderWidth = 8;
frame.Add (MainPanel);
MainPanel.PackStart(new Label("Nombre de la funcion"), false, false, 0);
MainPanel.PackStart(txtNameFunction, false, false, 0);
MainPanel.PackStart(new Label("Si recibe un argumento de tipo varchar")
    , false, false, 0);
MainPanel.PackStart(txtArg, false, false, 0);
Button btnSubmit = new Button("Ejecutar funcion");
btnSubmit.Clicked += new EventHandler(btnSubmitClicked);
MainPanel.PackStart(btnSubmit, false, false, 0);
MainPanel.PackStart(lbMsg, false, false, 0);
lbMsg.LineWrap = true;
ShowAll();
}

public void OnWindowDelete(object o, DeleteEventArgs args) {
Application.Quit();
}

void btnSubmitClicked(object o, EventArgs args){
string cStr = "Server=127.0.0.1;Port=5432;User Id=postgres;" +
    "Password=postgres;Database=pubs;";
try{
using(NpgsqlConnection conn = new NpgsqlConnection(cStr))
{
conn.Open();
NpgsqlCommand cmd = new NpgsqlCommand(txtNameFunction.Text, conn);
cmd.CommandType = System.Data.CommandType.StoredProcedure;
//en caso de tener un parametro como en nuestra funcion de ejemplo
if(txtArg.Text.Length > 0){
cmd.Parameters.Add(new NpgsqlParameter());
cmd.Parameters[0].NpgsqlDbType = NpgsqlDbType.Varchar;
cmd.Parameters[0].Value = txtArg.Text;
}
lbMsg.Text += cmd.ExecuteScalar().ToString();
}}catch(Exception e){
lbMsg.Text += e.Message;
}
}

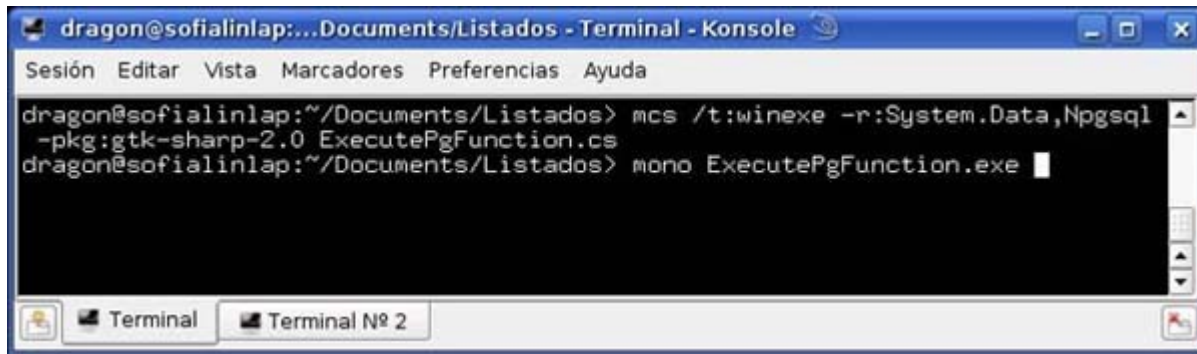
static void Main(string[] args) {
Application.Init();
new PgForm();
Application.Run();
}}
}

```

Lo compilamos y lo ejecutamos

```
mcs /t:winexe -r:System.Data,Npgsql -pkg:gtk-sharp-2.0 ExecutePgFunction.cs
```

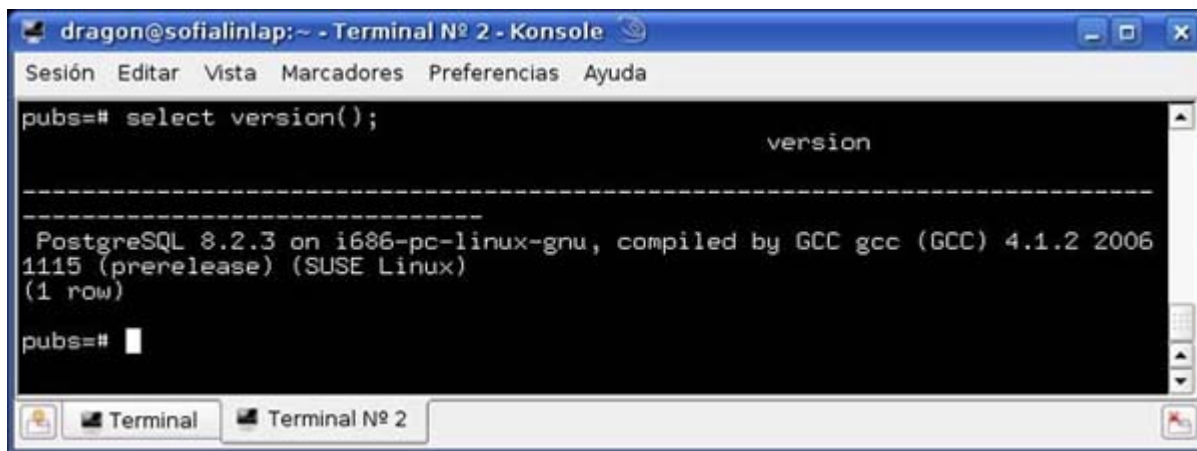
Fig 5 Compilación del programa



```
dragon@sofialinlap:~/Documents/Listados - Terminal - Konsole
Sesión Editar Vista Marcadores Preferencias Ayuda
dragon@sofialinlap:~/Documents/Listados> mcs /t:winexe -r:System.Data,Npgsql
-pkg:gtk-sharp-2.0 ExecutePgFunction.cs
dragon@sofialinlap:~/Documents/Listados> mono ExecutePgFunction.exe
```

Podemos probar nuestro programa invocando la función `version()` predeterminada de PostgreSQL.

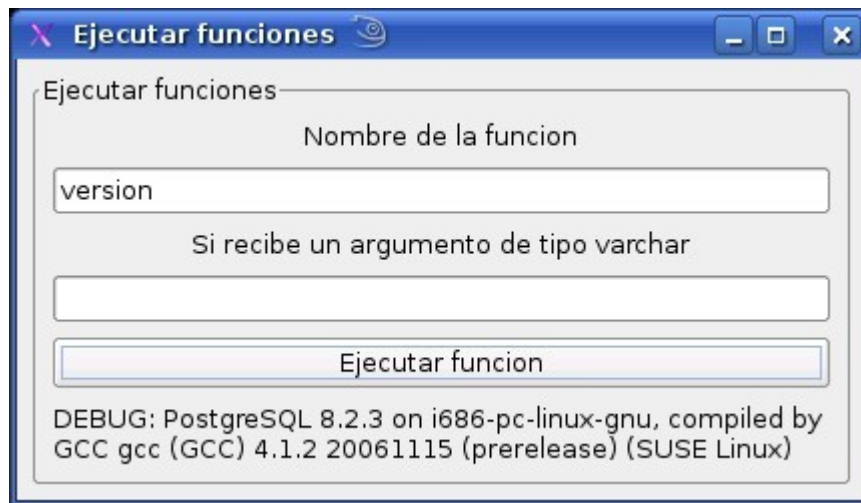
Fig 6 Ejecutando la función `version()` de PostgreSQL



```
dragon@sofialinlap:~ - Terminal Nº 2 - Konsole
Sesión Editar Vista Marcadores Preferencias Ayuda
pubs=# select version();
               version
-----
 PostgreSQL 8.2.3 on i686-pc-linux-gnu, compiled by GCC gcc (GCC) 4.1.2 2006
1115 (prerelease) (SUSE Linux)
(1 row)
pubs=#
```

Al ejecutar la función sin argumentos desde el formulario se vera el mismo resultado.

Fig 7 El programa ejecutando función version().



Aquí el driver de PostgreSQL para .NET ejecuta la función usando la clase *NpgsqlCommand* la cual recibe como argumento el nombre de la función y la conexión al servidor donde se encuentra.

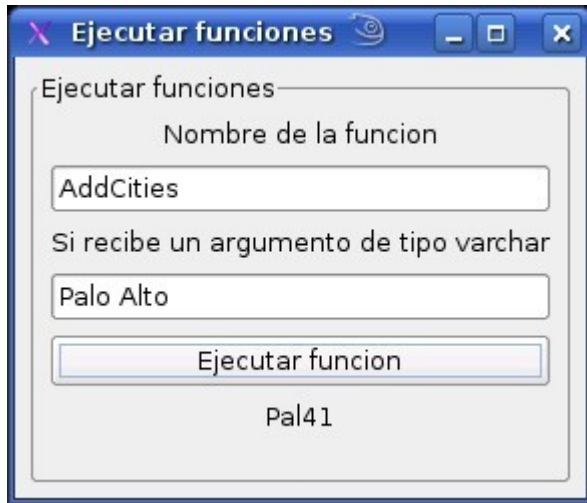
```
NpgsqlCommand cmd = new NpgsqlCommand("version", conn);
```

Si la función recibe parámetros, debemos de crear una instancia de la clase *NpgsqlParameter()* por cada uno de los parametros que reciba, es muy importante no olvidar indicarle a la clase *NpgsqlCommand* que el comando que ejecutaremos es un stored procedure o una función pl/sql, esto lo hacemos mediante la instrucción:

```
cmd.CommandType = System.Data.CommandType.StoredProcedure;
```

Para mayor referencia no olvidar leer la documentación del data provider para PostgreSQL. Si todo es ejecutado correctamente, ya podemos probar la función *AddCities* con el argumento del nombre de la ciudad y debe devolvernos la clave primaria de la tabla *cities*.

Fig 8 El programa ejecutando la función AddCities



[Descarga el código fuente](#)

Este documento está protegido bajo la licencia de documentación libre *Free Documentacion License* del Proyecto GNU, para consulta ver el sitio <http://www.gnu.org/licenses/fdl.txt> , toda persona que lo desee está autorizada a usar, copiar y modificar este documento según los puntos establecidos en la «Licencia FDL»