

Trabajando con Store Procedures PL/pgSQL en PostgreSQL

por Martín Márquez <xomalli@gmail.com>

Los lenguajes de procedimientos (procedural languages) han extendido la funcionalidad de las bases de datos proporcionando al lenguaje SQL cierto flujo de control similar al de un lenguaje de programación.

Cada fabricante tiene su propia implementación de un lenguaje de procedimiento basado en SQL, así **Microsoft SQL Server** tiene **Transact-SQL**, **Oracle** tiene **PL/SQL** y así sucesivamente una lista de bases de datos con su correspondiente lenguaje de procedimiento. En el caso de PostgreSQL se tienen varios lenguajes que pueden usarse como lenguajes de procedimiento entre ellos tenemos a PL/Tcl, PL/Perl, PL/Python, C y como opción predeterminada PL/pgSQL.

Las ventajas de ejecutar funciones del lado del servidor o Store Procedures con **PL/pgSQL** son las siguientes:

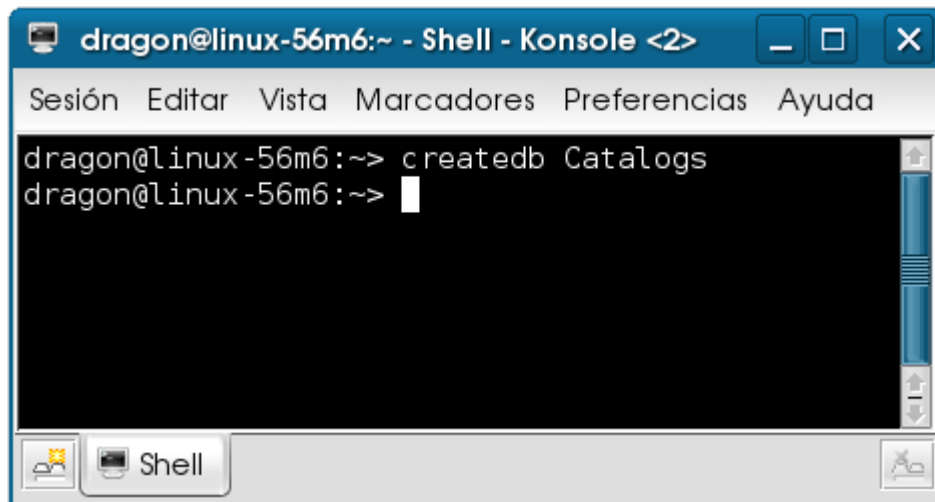
1. **Extensibilidad:** PL/pgsql es como un lenguaje de programación incluye la asignación y evaluación de variables y la posibilidad de hacer iteraciones y cálculos más complejos que con SQL.
2. **Seguridad:** Ayuda a evitar ciertos ataques con SQL Injection ya que al utilizar parámetros evita la construcción de comandos SQL utilizando la concatenación de cadenas.
3. **Rapidez:** Son ejecutados más rápidamente que las consultas SQL individuales ya que después de la primera ejecución el plan de ejecución se mantiene en memoria y el código no tiene que ser analizado ni optimizado nuevamente.
4. **Programación modular:** similar a los procedimientos y funciones en los lenguajes de programación, lo que evita que se tengan planas de sentencias SQL.
5. **Reutilización:** de código una función puede ejecutarse dentro de distintos Store Procedures.
6. **Rendimiento:** un Store Procedure puede contener decenas de sentencias SQL que son ejecutadas como una sola unidad de golpe, a diferencia de las sentencias SQL individuales donde hay que esperar a que cada una sea procesada.

Mediante los siguientes ejemplos mostraremos el uso de Store Procedures utilizando **PL/pgSQL**.

Creamos una base de datos de ejemplo llamada Catalogs con el siguiente comando desde el Shell.

```
$ createdb Catalogs
```

Fig 1. Creando la base de datos de ejemplo

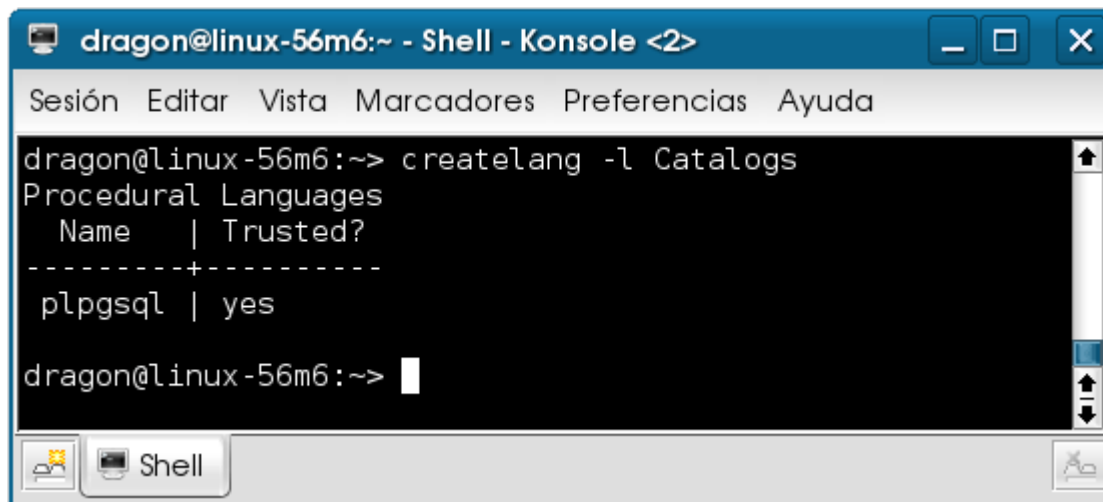


```
dragon@linux-56m6:~ - Shell - Konsole <2>
Sesión Editar Vista Marcadores Preferencias Ayuda
dragon@linux-56m6:~> createdb Catalogs
dragon@linux-56m6:~> 
```

Revisamos los lenguajes de procedimiento instalados en la base de datos, revisamos que **PL/pgSQL** se encuentre instalado con el siguiente comando.

```
$ createlang -l Catalogs
```

Fig 2. Revisando la instalación del lenguaje PLSQL



```
dragon@linux-56m6:~ - Shell - Konsole <2>
Sesión Editar Vista Marcadores Preferencias Ayuda
dragon@linux-56m6:~> createlang -l Catalogs
Procedural Languages
  Name   | Trusted?
-----+-----
 plpgsql | yes
dragon@linux-56m6:~> 
```

Si no se encuentra, entonces ejecutamos el siguiente comando para instalarlo, esto siempre como administrador del servidor:

```
$ createlang -U postgres plpgsql Catalogs
```

Si está instalado entonces abrimos un editor de texto y creamos un archivo llamado catalogs.sql donde escribiremos los comandos para crear las tablas de ejemplo y los Store Procedures para administrar los registros de cada una de las tablas.

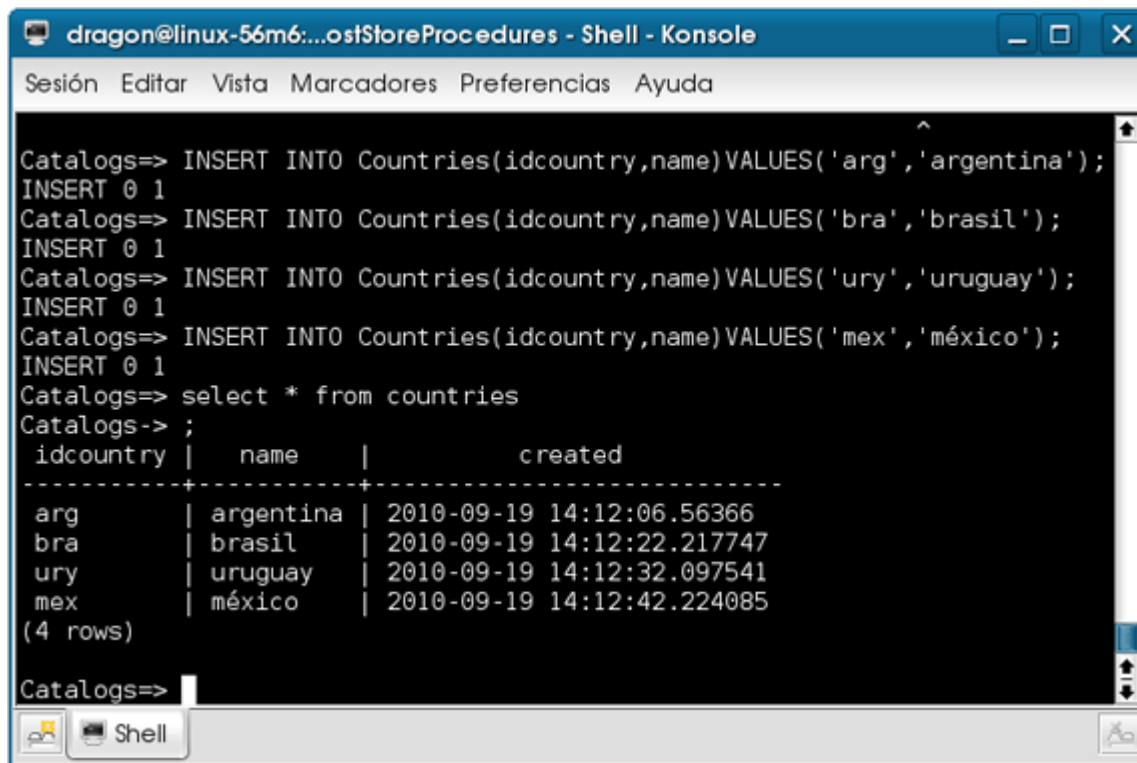
Para la creación de las tablas escribimos lo siguiente:

```
CREATE TABLE Countries(  
idcountry char(3) NOT NULL PRIMARY KEY  
,name varchar(250) NOT NULL  
,created timestamp DEFAULT NOW() NOT NULL  
);  
  
CREATE TABLE States(  
idstate serial NOT NULL PRIMARY KEY  
,name varchar(250) NOT NULL  
,idcountry varchar(3) NOT NULL REFERENCES Countries(idcountry)  
,created timestamp DEFAULT NOW() NOT NULL  
);  
  
CREATE TABLE Cities(  
idcity serial NOT NULL PRIMARY KEY  
,name varchar(250) NOT NULL  
,idstate int NOT NULL REFERENCES States(idstate)  
,created timestamp DEFAULT NOW() NOT NULL  
);
```

En acuerdo con la llave foránea definida en cada tabla, debe existir un país para poder crear un estado, así mismo debe de existir un estado para poder crear una ciudad, entonces creamos unos registros en la tabla países

```
INSERT INTO Countries(idcountry,name)VALUES('arg','argentina');  
INSERT INTO Countries(idcountry,name)VALUES('bra','brasil');  
INSERT INTO Countries(idcountry,name)VALUES('ury','uruguay');  
INSERT INTO Countries(idcountry,name)VALUES('mex','mexico');
```

Fig 3. Insertando los registros en la tabla



The screenshot shows a terminal window titled "dragon@linux-56m6:...ostStoreProcedures - Shell - Konsole". The terminal displays a series of SQL commands and their outputs. The commands are: `INSERT INTO Countries(idcountry,name)VALUES('arg','argentina');`, `INSERT INTO Countries(idcountry,name)VALUES('bra','brasil');`, `INSERT INTO Countries(idcountry,name)VALUES('ury','uruguay');`, and `INSERT INTO Countries(idcountry,name)VALUES('mex','m xico');`. Each insert command is followed by the output `INSERT 0 1`. Finally, the command `select * from countries` is executed, resulting in a table with 4 rows. The table has columns `idcountry`, `name`, and `created`. The data rows are: `arg | argentina | 2010-09-19 14:12:06.56366`, `bra | brasil | 2010-09-19 14:12:22.217747`, `ury | uruguay | 2010-09-19 14:12:32.097541`, and `mex | m xico | 2010-09-19 14:12:42.224085`. The terminal also shows the prompt `Catalogs=>` and `Catalogs->`.

```
Catalogs=> INSERT INTO Countries(idcountry,name)VALUES('arg','argentina');
INSERT 0 1
Catalogs=> INSERT INTO Countries(idcountry,name)VALUES('bra','brasil');
INSERT 0 1
Catalogs=> INSERT INTO Countries(idcountry,name)VALUES('ury','uruguay');
INSERT 0 1
Catalogs=> INSERT INTO Countries(idcountry,name)VALUES('mex','m xico');
INSERT 0 1
Catalogs=> select * from countries
Catalogs-> ;
 idcountry | name      | created
-----
 arg      | argentina | 2010-09-19 14:12:06.56366
 bra      | brasil    | 2010-09-19 14:12:22.217747
 ury      | uruguay   | 2010-09-19 14:12:32.097541
 mex      | m xico    | 2010-09-19 14:12:42.224085
(4 rows)

Catalogs=>
```

Supongamos que estas tablas van a utilizarse en un sistema donde sea obligatorio que los nombres de pa s, estado y ciudad, se almacenen teniendo la primera letra may scula o en notaci n Camel Case,(en este caso los pa ses quedaron guardados con letras min sculas de manera intencional, con el fin de mostrar un Store Procedure).

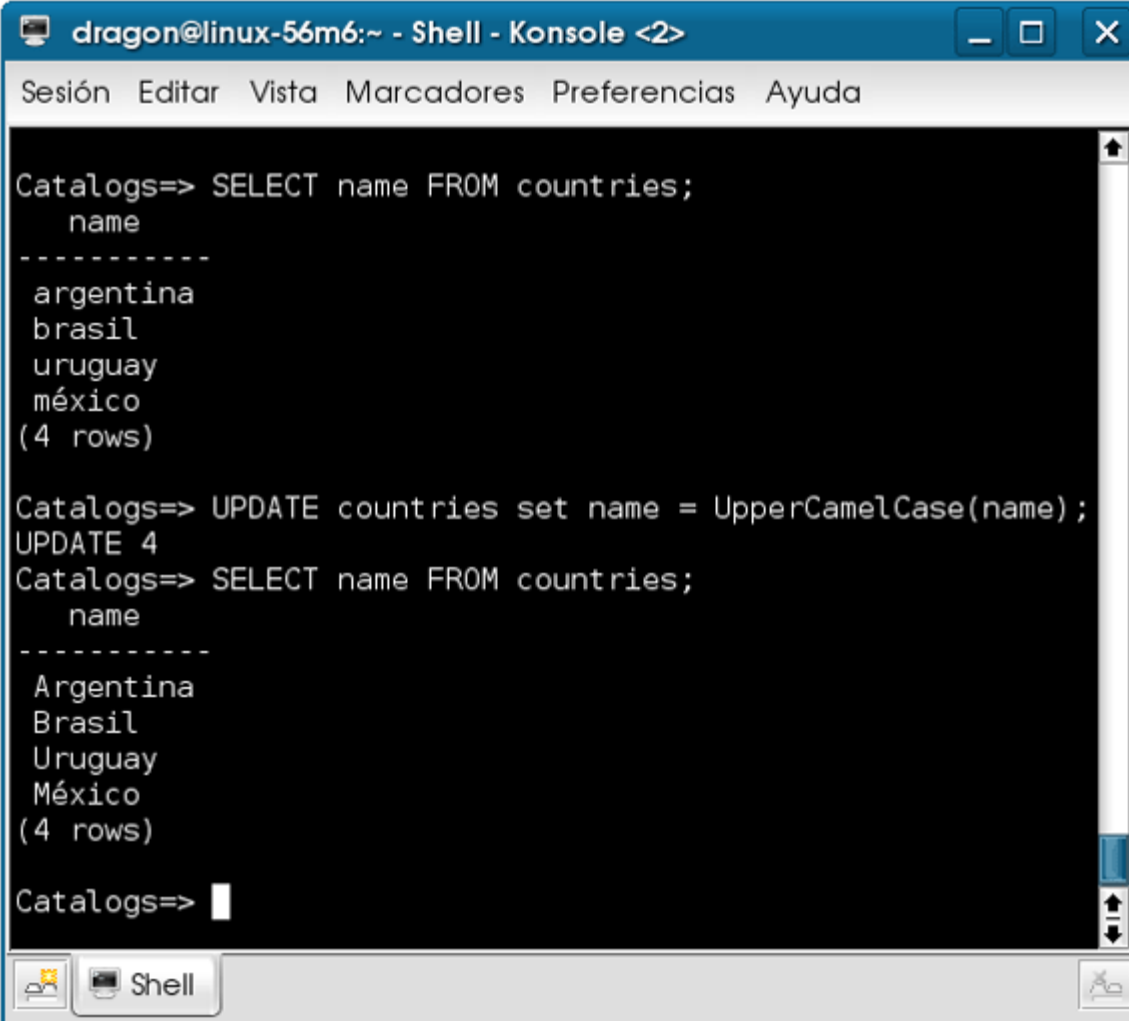
Creamos entonces una funci n para aplicar esta regla a los datos de la tabla countries.

La funci n se define con el siguiente c digo:

```
CREATE FUNCTION UpperCamelCase(varchar) RETURNS varchar AS
'DECLARE
  res varchar;
BEGIN
  SELECT INTO res UPPER(SUBSTRING($1,1,1)) || LOWER(SUBSTRING($1,2));
  return res;
END;'
LANGUAGE 'plpgsql';
```

Lo aplicamos de la siguiente manera para actualizar los registros.

Fig 4. Actualizando los registros utilizando la función *UpperCamelCase*



The image shows a terminal window titled "dragon@linux-56m6:~ - Shell - Konsole <2>". The window has a menu bar with "Sesión", "Editar", "Vista", "Marcadores", "Preferencias", and "Ayuda". The terminal content shows the following sequence of commands and output:

```
Catalogs=> SELECT name FROM countries;
  name
-----
argentina
brasil
uruguay
méxico
(4 rows)

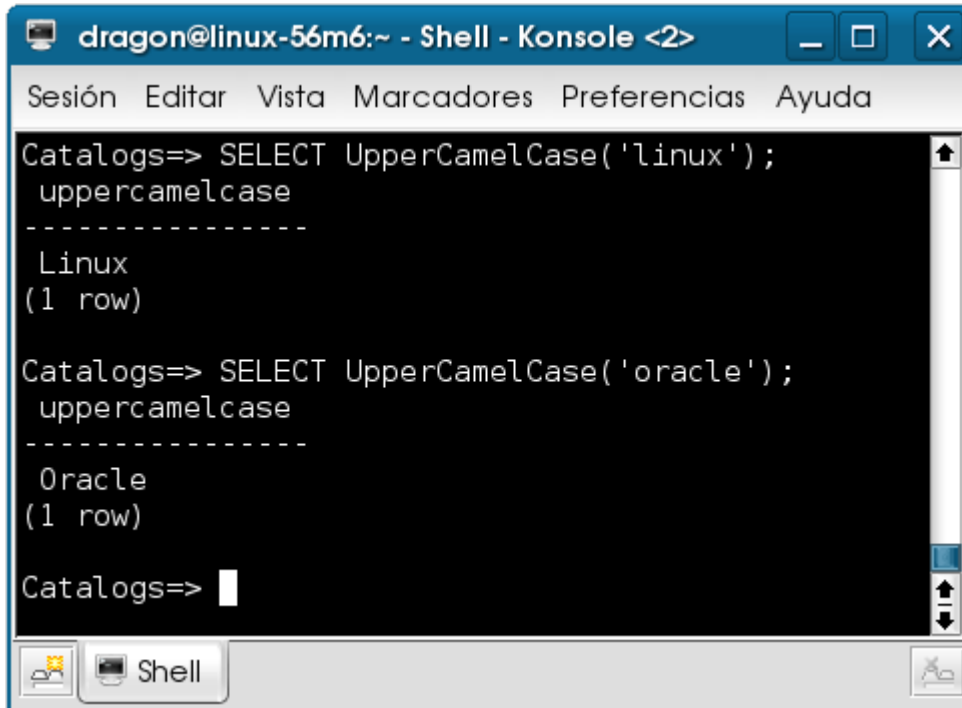
Catalogs=> UPDATE countries set name = UpperCamelCase(name);
UPDATE 4
Catalogs=> SELECT name FROM countries;
  name
-----
Argentina
Brasil
Uruguay
México
(4 rows)

Catalogs=> 
```

The terminal window includes standard window controls (minimize, maximize, close) in the top right corner and a status bar at the bottom with a "Shell" tab and a "Konsole" icon.

Una vez creada en el servidor se encuentra disponible para cualquier transformación que queramos aplicar sobre cualquier cadena.

Fig 5. Ejecutando la función *UpperCamelCase*



The screenshot shows a terminal window titled "dragon@linux-56m6:~ - Shell - Konsole <2>". The window has a menu bar with "Sesión", "Editar", "Vista", "Marcadores", "Preferencias", and "Ayuda". The terminal content shows two SQL queries being executed. The first query is "Catalogs=> SELECT UpperCamelCase('linux');", which returns a result set with one row containing "Linux". The second query is "Catalogs=> SELECT UpperCamelCase('oracle');", which returns a result set with one row containing "Oracle". The terminal window has a status bar at the bottom with icons for a shell and a file manager.

```
dragon@linux-56m6:~ - Shell - Konsole <2>
Sesión  Editar  Vista  Marcadores  Preferencias  Ayuda

Catalogs=> SELECT UpperCamelCase('linux');
uppercamelcase
-----
Linux
(1 row)

Catalogs=> SELECT UpperCamelCase('oracle');
uppercamelcase
-----
Oracle
(1 row)

Catalogs=> 
```

Ahora escribiremos las funciones para insertar registros en cada una de las tablas.

```
CREATE FUNCTION InsertState(varchar,varchar) RETURNS int AS
'DECLARE
  regs record;
  res numeric;
BEGIN
  SELECT INTO regs count(name) FROM States WHERE name = UpperCamelCase($1);
  IF regs.count = 0
  THEN
    INSERT INTO States(name,idcountry)VALUES(UpperCamelCase($1),$2);
    SELECT INTO res idstate FROM States WHERE name = UpperCamelCase($1);
    RETURN res;
  END IF;
  IF regs.count > 0
  THEN
    SELECT INTO res idstate FROM States WHERE name = UpperCamelCase($1);
    RETURN res;
  END IF;
END;
' LANGUAGE 'plpgsql';

CREATE FUNCTION InsertCity (varchar,numeric) RETURNS int AS
'DECLARE
  regs record;
  res numeric;
BEGIN
  SELECT INTO regs count(name) FROM Cities WHERE name = UpperCamelCase($1);
```

```

IF regs.count = 0
THEN
  INSERT INTO Cities(name,idstate)VALUES(UpperCamelCase($1),$2);
  SELECT INTO res idcity FROM Cities WHERE name = UpperCamelCase($1);
  RETURN res;
END IF;
IF regs.count > 0
THEN
  SELECT INTO res idcity FROM Cities WHERE name = UpperCamelCase($1);
  RETURN res;
END IF;
END;
' LANGUAGE 'plpgsql';

```

Básicamente la estructura de un Store Procedure es la siguiente:

```

CREATE FUNCTION [nombre de la función] ([parámetros separados por comas]) RETURNS
[el tipo de dato que regresa] AS
'DECLARE aquí se definen las variables que se usarán.
BEGIN indica el inicio de la función.
SELECT INTO este comando permite que los resultados de las consultas sean asignados
a variables.
(no debe de confundirse con SELECT [columnas] INTO)
RETURN Sale de la función y regresa el tipo de dato que se declaro después de la
palabra RETURNS del CREATE FUNCTION
END indica el fin de la función
' LANGUAGE indica con que lenguaje esta escrita la función, puede ser un lenguaje
de procedimiento
(plpgsql) o de consulta (SQL).

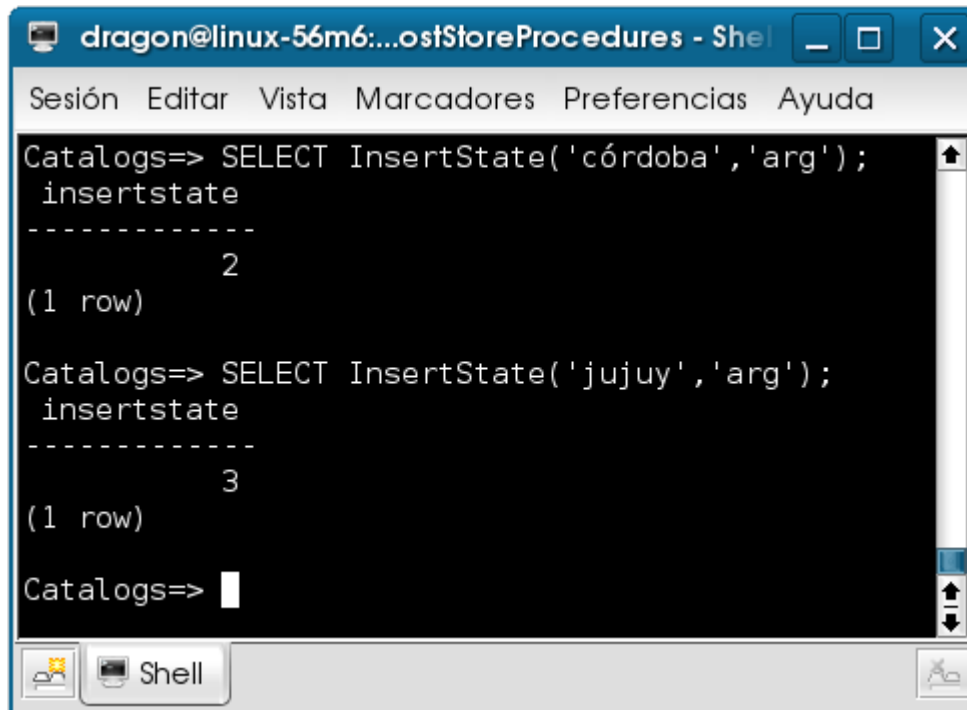
```

Vemos que en cada *Store Procedure*, reutiliza la función *UpperCamelCase(varchar)* que habíamos previamente creado.

Esto porque un *Store Procedure* puede llamar a otro *Store Procedure* que se encuentre disponible.

En las siguientes imágenes los resultados de la ejecución de cada *Store Procedure*.

Fig 6. Ejecución del procedimiento *InsertState*



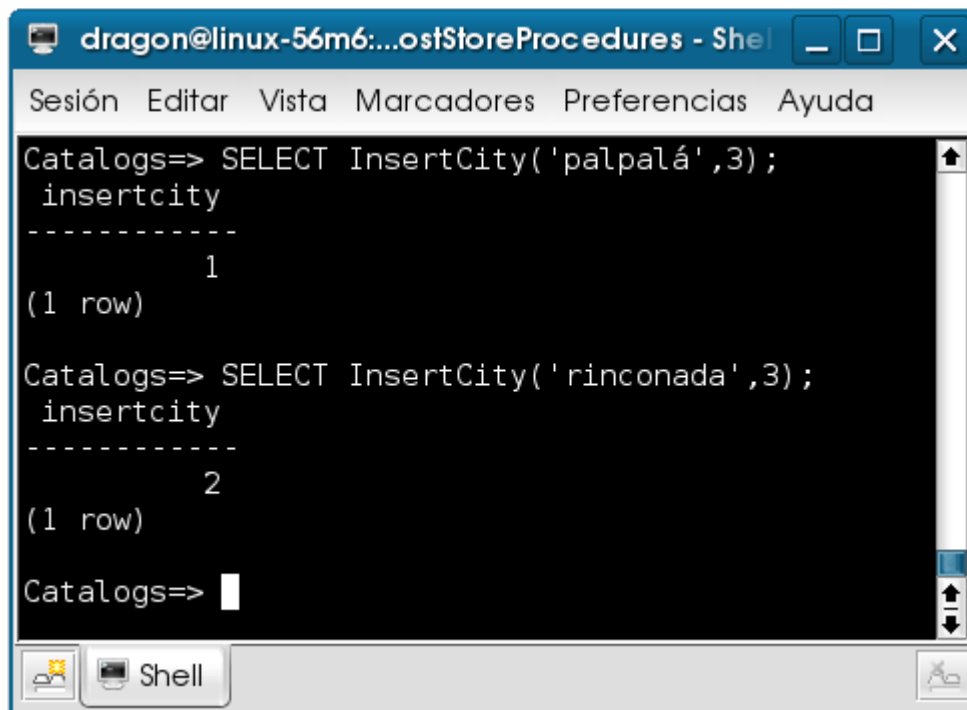
The screenshot shows a terminal window titled "dragon@linux-56m6:...ostStoreProcedures - Shell". The window has a menu bar with "Sesión", "Editar", "Vista", "Marcadores", "Preferencias", and "Ayuda". The terminal content shows two SQL queries being executed. The first query is "SELECT InsertState('córdoba','arg'); insertstate", which returns a single row with the value "2". The second query is "SELECT InsertState('jujuy','arg'); insertstate", which returns a single row with the value "3". The prompt "Catalogs=>" is visible at the bottom of the terminal.

```
dragon@linux-56m6:...ostStoreProcedures - Shell
Sesión  Editar  Vista  Marcadores  Preferencias  Ayuda
Catalogs=> SELECT InsertState('córdoba','arg');
insertstate
-----
                2
(1 row)

Catalogs=> SELECT InsertState('jujuy','arg');
insertstate
-----
                3
(1 row)

Catalogs=> 
```

Fig 7. Ejecución del procedimiento *InsertCity*



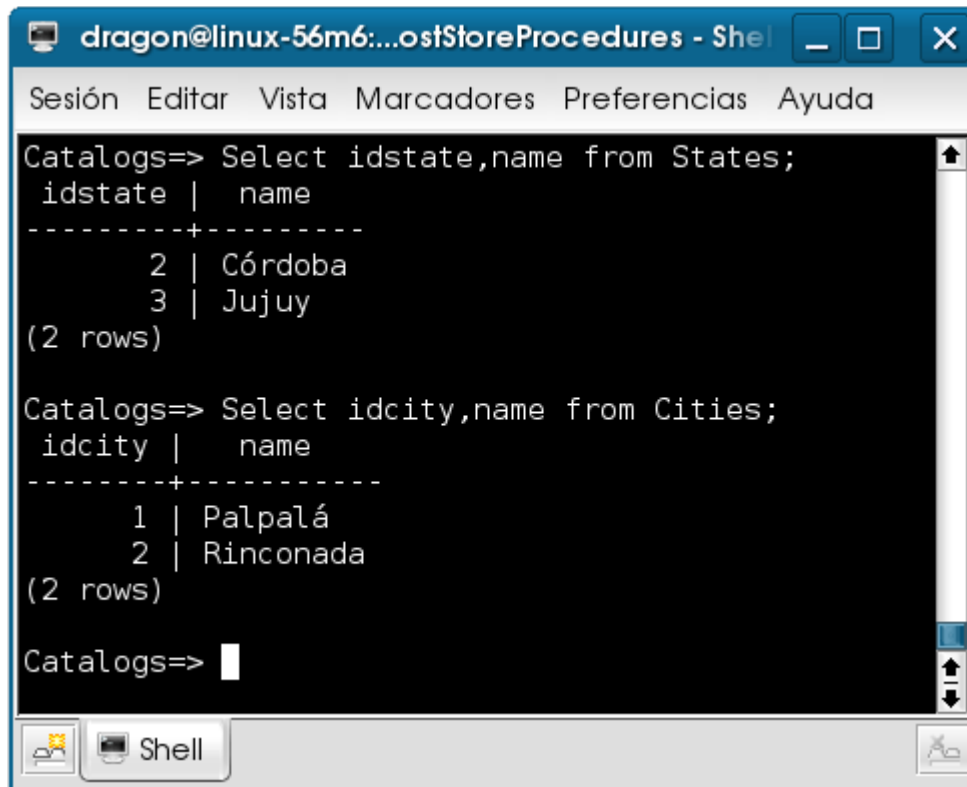
The screenshot shows a terminal window titled "dragon@linux-56m6:...ostStoreProcedures - Shell". The window has a menu bar with "Sesión", "Editar", "Vista", "Marcadores", "Preferencias", and "Ayuda". The terminal content shows two SQL queries being executed. The first query is "SELECT InsertCity('palpalá',3); insertcity", which returns a single row with the value "1". The second query is "SELECT InsertCity('rinconada',3); insertcity", which returns a single row with the value "2". The prompt "Catalogs=>" is visible at the bottom of the terminal.

```
dragon@linux-56m6:...ostStoreProcedures - Shell
Sesión  Editar  Vista  Marcadores  Preferencias  Ayuda
Catalogs=> SELECT InsertCity('palpalá',3);
insertcity
-----
                1
(1 row)

Catalogs=> SELECT InsertCity('rinconada',3);
insertcity
-----
                2
(1 row)

Catalogs=> 
```


Fig 8. Comprobando la aplicación de los procedimientos en los datos



```
dragon@linux-56m6:...ostStoreProcedures - Shell
Sesión  Editar  Vista  Marcadores  Preferencias  Ayuda

Catalogs=> Select idstate,name from States;
 idstate | name
-----+-----
         2 | Córdoba
         3 | Jujuy
(2 rows)

Catalogs=> Select idcity,name from Cities;
 idcity | name
-----+-----
        1 | Palpalá
        2 | Rinconada
(2 rows)

Catalogs=> 
```



[Descarga el código fuente para PostgreSQL](#)

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»