IIC2413 — Bases de Datos

PL/pqSQL

La mayoría de los DBMS ofrecen la posibilidad de programar funciones o procedimientos almacenados en el mismo DBMS, de forma que al correr estos se pueda tomar ventaja de todas las bondades de los DBMS. Esta semana vamos a aprender a programar estos procedimientos. Sqlite3 no soporta esta funcionalidad, así que vamos a tener que recurrir nuevamente a PostgeSQL.
Para minimizar la carga del servidor, asignaremos una base de datos nueva por cada grupo del proyecto. Para entrar a esta base de datos, ingresa vía ssh con tu nombre de usuario `grupoxx` como siempre. Para entrar a la base de datos, ejecuta:

psql grupoxxstored

Donde grupoxx es tu nombre de grupo. Esto te va a dejar ingresar a una nueva base de datos, llamada grupoxxstored.
Antes de comenzar, crea una relación

Personas(run:varchar, nombre:varchar,apellido:varchar)

Ahora estamos listos para crear nuestra primera función!

1. Funciones Básicas

La forma básica de las funciones es:

```
CREATE or REPLACE Function <nombre_función> (<atributos de input>)
RETURNS <tipo_retorno> AS
$$
DECLARE
    <declaracion de variables>
BEGIN
    <sentencias SQL>
END
$$ language plpqsql;
```

Acá tenemos:

- `<nombre_función>` Es el nombre que le quieras dar a la función

- `<atributos de input>` Es una lista de atributos que recibe la función como input, de acuerdo a la sintáxis `(input_1 tipo_1, input_2 tipo_2, . . . , input_n tipo_n)`
- `<declaración de variables>` Es una lista de variables a declarar. Los tipos más comunes son integer, numeric, varchar y record.
- `<sentencias SQL>` contiene una lista de instrucciones SQL, cada una terminando en punto y coma (;)

Hagamos entonces nuestra primera función. Abre en tu computador un archivo de texto, y copia

```
CREATE OR REPLACE FUNCTION
insertar_persona (rut varchar, nombre varchar, apellido varchar)
RETURNS void AS $$
BEGIN
insert into personas values (rut,nombre,apellido);
END
$$ language plpgsql
```

Guardalo como `insertar_persona.sql`. Ahora usa un servidor de ftp para mover este archivo al servidor bases.ing.puc.cl, a la carpeta home/grupoxx.
Vuelve a entrar a postgres, a la base de datos grupoxxstored. Ejecuta

\i insertar_persona.sql

Con esto le estás diciendo a postgreSQL que ejecute el comando que tenías guardado. PostgreSQL va crear la función, ¡que ya está lista para ser usada! Con esto podemos llenar datos de forma más rápida. Ejecuta

SELECT insertar_persona('unrut','unnombre','unapellido')

Y verifica que la inserción fue realizada revisando la tabla Persona.
¿Puedes entender que es lo que pasó y para qué sirve tu función?
Ahora intentemos algo más radical.

```
CREATE OR REPLACE FUNCTION
insercion_radical (numero integer)
RETURNS void AS $$
DECLARE
    temp varchar;
BEGIN
    FOR i IN 1..numero LOOP
        temp := to_char(i,'99999999');
        insert into personas values (temp,temp,temp);
    END LOOP;
END
$$ language plpgsql
```

Nuevamente, almacena esto como insercion_radical.sql. Llévalo al servidor con ftp y ejecuta en postgreSQL:

\i insercion_radical.sql

Ahora estamos listos! Ejecuta lo siguiente para tener tu primera tabla con 10.000 tuplas

SELECT insercion_radical(10.000)

Hay hartas cosas que explicar acá.

- El control de flujo en esta función está dado por

```
FOR <var> IN <x>..<y> LOOP
    <sentencias SQL>
END LOOP;
```

  No necesitas haber declarado `<var>`, pero solo es válida dentro del loop (de hecho, al entrar al loop ignoras la declaración anterior, si existiese). En general `<x>` e `<y>` pueden ser números, variables numéricas o expresiones.

- Existen otras formas de iteración, como `WHILE` y `LOOP`

- La línea `temp := to_char(i,'99999999');` dice que la variable `temp` ahora corresponde a lo que retorna la función `to\_char`. Esta función toma un número n y un string f , y devuelve el número n como un string, en el formato que indicaste con f. En general postgreSQL tiene miles de funciones como esta, que ya están hechas. ¡Consulta la documentación!

Como control de flujo puedes escribir

```
IF <condicion booleana> THEN
    <sentencias SQL>
ELSE
    <sentencias SQL>
END IF;
```

En este caso la condición booleana es cualquier comparación que puedas escribir en SQL.

1.1. Recorriendo los resultados de una consulta

La capacidad para recorrer, en el entorno mismo del DBMS, los resultados de las consultas es quizá lo más importante en las funciones.
Para eso necesitamos una variable de tipo `RECORD`. Este es un tipo abstracto que sirve para contener los resultados de una tupla.
La forma básica de recorrer los resultados de una consulta es entonces:

```
FOR <record> IN <consulta SQL> LOOP
    <sentencias SQL>
END LOOP;
```

Esto funciona de la siguiente manera: El sistema ejecuta la `<consulta SQL>`, y va iterando tupla a tupla la respuesta: la primera tupla de la respuesta queda guardada en la variable `<record>`, cuando se terminan de ejecutar las `<sentencias SQL>` se pasa a la segunda iteración, donde la segunda tupla de la respuesta a `<consulta SQL>` pasa a la variable `<record>`, y así sucesivamente.
A modo de ejemplo, crea una tabla en postgres

PersonasCompleto(run:varchar, nombrecompleto:varchar)

La siguiente función se usa para copiar a PersonasCompleto el rut y la concatenación del nombre y el apellido de cada persona almacenado en la tabla Persona

```
CREATE OR REPLACE FUNCTION
transferencia ()
RETURNS void AS $$
DECLARE
    tupla RECORD;
    concat varchar;
BEGIN
    FOR tupla IN SELECT * FROM Personas LOOP
        concat = tupla.nombre || tupla.apellido;
        insert into personascompleto values (tupla.rut, concat);
    END LOOP;
END
$$ language plpgsql
```

Ejercicios:

- Escribe una función que reciba un número i y retorne la i-esima potencia de 2 (no tiene que ver con base de datos, pero es bueno para revisar que entiendes el entorno de programación!)
- Escribe una función que retorne el número de tuplas en la tabla Personas.
- Escribe una función `contar_K()` que entregue el número de ruts en la tabla Personas que terminan con K (recuerda que puedes usar el operador `LIKE`).