Práctica 11

Procedimientos Almacenados

1. Objetivo General

Conocer los elementos básicos de un procedimiento almacenado.

2. Objetivos Secundarios

- Ampliar la funcionalidad de un SMBD.
- Automatizar procesos en un SMBD.

3. Introducción

Los SMBD como sabemos, almacenan tablas que contienen datos y cuentan con la capacidad de manejar y consultar estos datos a través de lenguajes como SQL. Los códigos que resultan de las consultas de éste manejo de datos pueden ser extensos y complejos debido a la naturaleza de cada problema en particular.

Si suponemos que una actividad recurrente para los usuarios de la base de datos utiliza una de estas complicadas combinaciones de consultas o funciones, y esto resulta en un aumento considerable del tiempo para escribir una y otra vez el código, resulta razonable pensar en una manera de automatizar dichos procesos. Para estos casos, un procedimiento almacenado o Stored Procedure (SP) se presenta como una solución.

Los SPs son funciones implementadas que se almacenan físicamente en el SMBD y que en consecuencia tienen acceso directo a los datos. Siendo esta una ventaja con respecto a funciones similares pero implementadas de manera ajena al SMBD con otros lenguajes de programación.

Los SPs se implementan utilizando PL/pgSQL, este lenguaje de programación imperativo, provisto por PostgreSQL, permite la construcción de funciones más complejas al incluir validaciones, cálculos y procedimientos integrados con sentencias SQL. Los SPs cuentan con la estructura de un programa al cuál se le pueden definir parámetros de entrada y salida, constantes, reglas de validación y por supuesto sentencias SQL de cualquier tipo.

3.1. Estructura de un SP

Al ser un programa, los SPs obedecen cierta sintaxis que varía de acuerdo al SMBD que se ocupe. Para esta práctica se estudiará la estructura de la sintaxis para PostgreSQL.

La Figura 11.1 muestra los diferentes bloques de los que se compone la sintaxis de un SP en PostgreSQL.

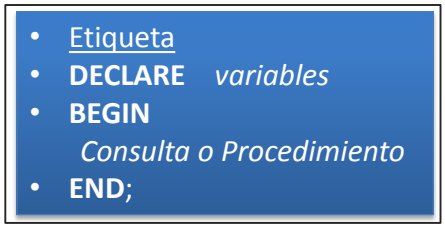

Figura 11.1 - Bloques de un SP en PostgreSQL.

Cada bloque ofrece características distintas para el correcto desarrollo del proceso que se pretende definir. En la siguientes subsecciones se detallarán los contenidos de cada uno de estos bloques.

3.1.1. Etiqueta

Es el primer bloque del SP y es aquí donde se define el nombre de la función, los parámetros de entrada, es decir, los que ingresará el usuario, y el tipo de dato del parámetro de salida de la función. Ver Figura 11.2.

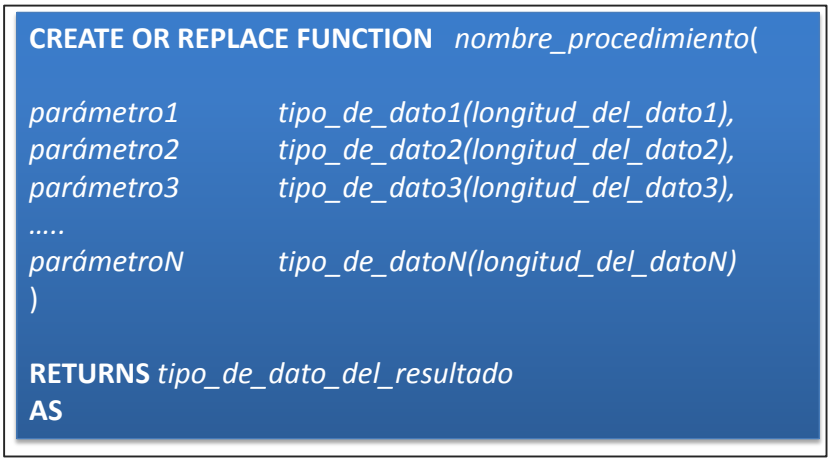

Figura 11.2 - Sintaxis de la etiqueta de un SP.

En esta sintaxis encontramos:

1. CREATE OR REPLACE FUNCTION
   Palabras reservadas para dar comienzo a la definición de la función. Cabe destacar que la frase OR REPLACE es una opción que permite redefinir una función creada anteriormente con el mismo nombre, si esta existiera, por la que se definirá en ese momento. Su funcionalidad se presenta cuando se está modificando constantemente el código o cuando se encuentra en la fase de implementación y prueba. No es necesario escribirlo.
2. nombre_procedimiento
   Es el nombre que se le dará al procedimiento.
3. nombre_parámetro1
   Es el nombre con el que se define a un parámetro de entrada.

4. tipo_de_dato_1
   Es el tipo de dato que está relacionado con el parámetro de entrada.
5. (longitud_del_tipo_de_dato_variable1)
   Es la longitud del parámetro. Recordando que existen tipos de datos soportados en PostgreSQL que no necesitan la definición de la longitud, como el caso de integer.
6. RETURNS
   Palabra reservada para definir cuál será la salida o el resultado que se desea del SP. Si se espera más de un valor, es decir un conjunto de valores, se utiliza la opción RETURNS SETOF y el tipo de dato. En caso de que se espere una tabla como resultado la opción a utilizar es RETURNS TABLE y como tipo de dato se escribirán el nombre_columna y el tipo_de_dato de cada columna que componga la tabla resultante.
7. tipo_de_dato_del_resultado
   Define el tipo del resultado que se espera.
8. AS
   Palabra reserva para indicar el término de la etiqueta y marcar el comienzo del cuerpo de la función. El cuerpo de la función es donde se definirán las variables y la operación formal del proceso.

3.1.2. DECLARE

Dentro de este bloque se definirán todas las variables que sean necesarias para el desarrollo del proceso que se implementará. En la Figura 11.3 se mostrará la sintaxis del bloque de declaración de variables.

```
$$
DECLARE nombre_variable1  tipo_de_dato_variable1 (longitud_del_tipo_de_dato_variable1);
DECLARE nombre_variable2  tipo_de_dato_variable2 (longitud_del_tipo_de_dato_variable2);
DECLARE nombre_variable3  tipo_de_dato_variable3 (longitud_del_tipo_de_dato_variable3);
....
DECLARE nombre_variableN  tipo_de_dato_variableN (longitud_del_tipo_de_dato_variableN);
```

Figura 11.3 - Sintaxis del bloque de declaraciones de variables.

En esta sintaxis encontramos:

1. $$
   Los símbolos de moneda indican el comienzo del cuerpo de la función, es decir, las operaciones necesarias para conseguir el resultado esperado. Estos símbolos son la contracción de $BODY$ y se escribe al comienzo de la definición así como al final de la misma.
2. DECLARE
   Palabra reservada para dar inicio a la declaración de una variable. Esta palabra reservada se deberá escribir cada que se requiera definir una variable.

3. nombre_variable1
   Es el nombre con el que se define a una variable.
4. tipo_de_dato_variable1
   Es el tipo de dato que está relacionado con la variable1.
5. (longitud_del_tipo_de_dato_variable1)
   Es la longitud de la variable.

3.1.3. BEGIN y END

En este bloque se define el proceso de manera secuencial, es decir, el orden en el que se coloque el conjunto de consultas será el mismo en el que se ejecutarán. Comienza con la palabra reservada BEGIN mientras que la finalización del proceso se indica con la palabra END.

A continuación la Figura 11.4 muestra el cuerpo de un procedimiento.

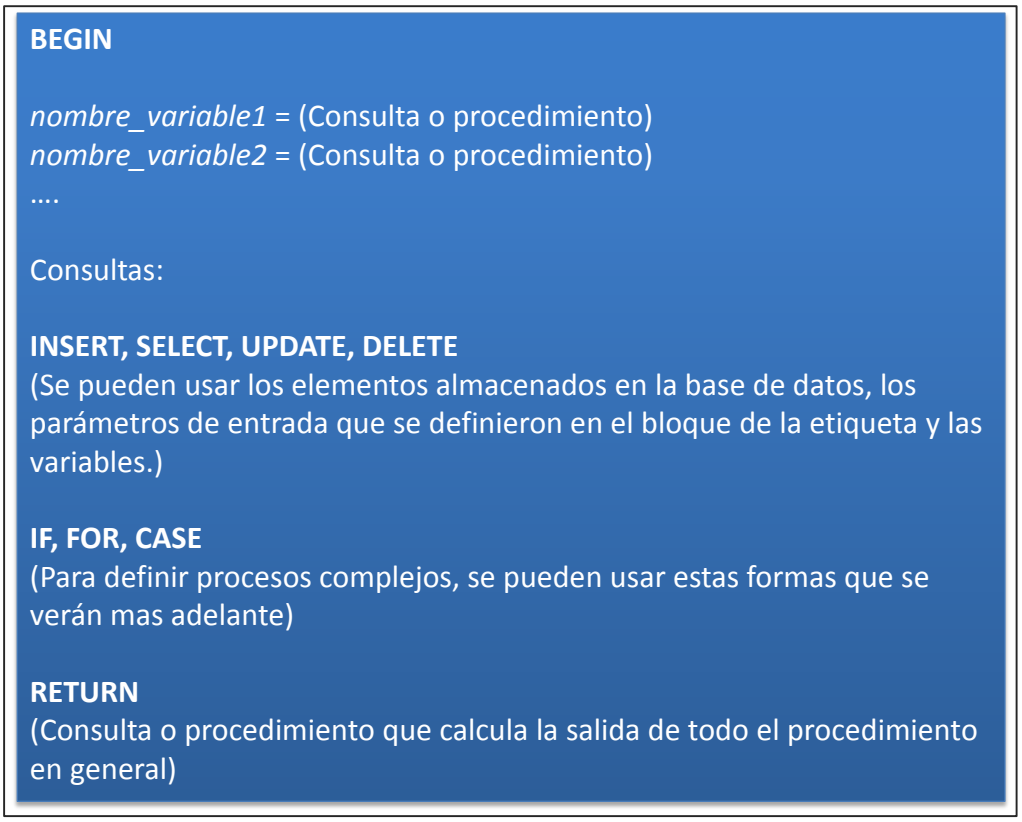

Figura 11.4 - Cuerpo de un procedimiento.

Para el cuerpo de un procedimiento hay que tomar en cuenta que el orden en que se definan los valores de las variables o las consultas depende de cómo será estructurado, es decir, si necesitamos el valor que toma una variable como parámetro dentro de una consulta, es necesario definirlo antes de la consulta. Si es independiente el valor de la variable a los parámetros de la consulta, el orden es indistinto.

En esta sintaxis encontramos:

1. BEGIN
   Palabra reservada para comenzar con la definición del cuerpo del procedimiento.
2. nombre_variable1 = (Consulta o procedimiento)
   Es la variable a la cual se le asignará un cierto valor por medio de consultas o procedimientos.
3. Consultas:
   No es palabra reservada. Se incluyó en la Figura 11.4 como indicativo de que el cuerpo de un procedimiento puede incluir consultas o modificaciones. También existen casos para los que las consultas más complejas necesitan apoyarse en sintaxis del tipo FOR para casos iterativos, IF para condicionales y CASE como una generalización para IF anidados. Las cuales se pueden definir en esta parte del SP.
4. RETURN
   Palabra reservada para comenzar con la definición de la constante, el proceso o la consulta que será el resultado de todo el proceso. Es usado para saber, entre otras cosas, si el proceso fue exitoso o no, o que camino tomó el procedimiento dependiendo de las condiciones de las consultas definidas.

Por último, la Figura 11.5 muestra la sintaxis para finalizar el cuerpo y la sintaxis completa del procedimiento.

Figura 11.5 - Bloque de término del Store Procedure.

En esta sintaxis encontramos:

1. END;
   Palabra reservada para definir el cierre del cuerpo del procedimiento, es decir, que el procedimiento ha finalizado. El símbolo de punto y coma indica al SMBD que ejecute todo lo definido en el cuerpo del procedimiento.
2. $$
   Los símbolos de moneda cierran los símbolos que se abrieron en la declaración de variables de la Figura 11.3.
3. LANGUAGE 'plpgsql'
   Define el lenguaje que se utilizó en el procedimiento antes definido. La cadena de caracteres 'plpgsql' denota el lenguaje procedural de PostgreSQL.

4. Ejemplos

A continuación se construirán una serie de ejemplos aumentando el número de consultas, variables y supuestos para insertar un nombre en la base de datos *Pizzeria*.

En la Figura 11.6 se muestra la sintaxis para insertar los datos de un nuevo cliente a la tabla *Cliente*. Los parámetros de entrada, o insumos del SP, son: nombre, apellido paterno y materno. No existen variables y el cuerpo del SP es solamente una sentencia de tipo INSERT.

```
CREATE OR REPLACE FUNCTION SP_Inserta_datos_nuevo_cliente
(nombre VARCHAR(50),
 ap_paterno VARCHAR(50),
 ap_materno VARCHAR(50))

RETURNS VARCHAR(20)
AS
$$
BEGIN

INSERT INTO Cliente (nombre_cliente, apellido_pat_cliente, apellido_mat_cliente)
VALUES (nombre, ap_paterno, ap_materno);

RETURN 'Insercion_Exitosa';

END;
$$
LANGUAGE 'plpgsql';
```

Figura 11.6 - Sintaxis ejemplo de inserción (1).

La tabla Cliente de la base de datos Pizzeria también incluye la columna id_Cliente la cual no se tomó en cuenta en el SP anterior. Por lo tanto haremos una consulta que calcule este valor y lo inserte en la tabla. Ver Figura 11.7.

```
CREATE OR REPLACE FUNCTION SP_Inserta_datos_nuevo_cliente2
(nombre VARCHAR(50),
ap_paterno VARCHAR(50),
ap_materno VARCHAR(50))

RETURNS VARCHAR(20)
AS
$$

DECLARE id_cliente_calculado INTEGER;

BEGIN

id_cliente_calculado = CASE WHEN((SELECT MAX(id_cliente)
                                        FROM Cliente) IS NOT NULL)
                            THEN (SELECT MAX(id_cliente)
                                    FROM Cliente)+1
                             ELSE 1
END;

INSERT INTO Cliente VALUES (id_cliente_calculado,nombre,ap_paterno,ap_materno);

RETURN 'Insercion_Exitosa';

END;
$$
LANGUAGE 'plpgsql';
```

Figura 11.7 - Sintaxis ejemplo de inserción (2).

El código de la Figura 11.7 ya puede poblar todas las columnas de la tabla Cliente, pero nada impide que se pueda insertar más de una vez el mismo nombre de Cliente lo cual derivaría en un error de integridad.

El siguiente código realizará una búsqueda en la base y si resulta que el nombre de cliente ya se encuentra en ésta tabla no lo insertará, en otro caso realizará la inserción. Ver Figura 11.8.

```
CREATE OR REPLACE FUNCTION SP_Inserta_datos_nuevo_cliente3
(nombre VARCHAR(50),
ap_paterno VARCHAR(50),
ap_materno VARCHAR(50))

RETURNS VARCHAR(20)
AS
$$

DECLARE id_cliente_calculado INTEGER;
DECLARE estatus_insercion VARCHAR(25);

BEGIN

IF ((SELECT count(*)
      FROM cliente
      WHERE nombre_cliente = nombre
      AND apellido_pat_cliente = ap_paterno
      AND apellido_mat_cliente = ap_materno) = 0)
THEN
  id_cliente_calculado = CASE
  WHEN((SELECT MAX(id_cliente)
       FROM Cliente) IS NOT NULL)
  THEN (SELECT MAX(id_cliente)
       FROM Cliente)+1
  ELSE 1
  END;

  INSERT INTO Cliente VALUES (id_cliente_calculado,nombre,ap_paterno,ap_materno);

  estatus_insercion = 'Insercion_Exitosa';

ELSE

estatus_insercion = 'El_cliente_ya_existe';

END IF;

RETURN estatus_insercion;

END;
$$
LANGUAGE 'plpgsql';
```

Figura 11.8 - Sintaxis ejemplo de inserción (3).

La base de datos Pizzeria almacena los datos necesarios para el buen funcionamiento de un negocio de elaboración de pizzas y reparto a domicilio de las mismas. Es por eso que resulta

importante almacenar los datos del domicilio y teléfono del cliente. Se necesita un SP que realice las inserciones pertinentes en todas las tablas necesarias para relacionar los datos.

Estas tablas son: Cliente, Direccion_Cliente y Telefono_Cliente. Los datos a insertar en cliente son los mismos que se incluyeron en el SP de la Figura 11.8. Los datos de la dirección son: calle, número exterior, número interior, colonia y delegación. Mientras que los del teléfono son: identificador del cliente y el número del teléfono.

```
CREATE OR REPLACE FUNCTION SP_Inserta_datos_nuevo_cliente4
(nombre VARCHAR(50),
 ap_paterno VARCHAR(50),
 ap_materno VARCHAR(50),
 telefono VARCHAR(20),
 dom_calle VARCHAR(60),
 dom_num_exterior INTEGER,
 dom_num_interior INTEGER,
 dom_colonia VARCHAR(70),
 dom_delegacion VARCHAR(70))

RETURNS VARCHAR(20)
AS
$$
DECLARE id_cliente_calculado INTEGER;
DECLARE estatus_insercion VARCHAR(25);
BEGIN
IF ((SELECT count(*)
      FROM cliente
      WHERE nombre_cliente = nombre
      AND apellido_pat_cliente = ap_paterno
      AND apellido_mat_cliente = ap_materno) = 0)
THEN
   id_cliente_calculado = CASE
   WHEN((SELECT MAX(id_cliente)
        FROM Cliente) IS NOT NULL)
   THEN (SELECT MAX(id_cliente)
        FROM Cliente)+1
   ELSE 1
   END;
   INSERT INTO Cliente VALUES (id_cliente_calculado,nombre,ap_paterno,ap_materno);
   INSERT INTO Direccion_Cliente
   VALUES
(id_cliente_calculado,dom_calle,dom_num_exterior,dom_num_interior,dom_colonia,dom_delegacion);
   INSERT INTO Telefono_Cliente VALUES (id_cliente_calculado,telefono);

   estatus_insercion = 'Insercion_Exitosa';
ELSE
estatus_insercion = 'El_cliente_ya_existe';
END IF;
RETURN estatus_insercion;
END;
$$
LANGUAGE 'plpgsql';
```

Figura 11.9 - Sintaxis ejemplo de inserción (4).

La Figura 11.9 muestra una sintaxis compleja para realizar la tarea de inserción en las tablas pertinentes a los datos del cliente. Si suponemos que esta actividad se realiza cientos de veces, el usuario que ejecute las consultas, estaría escribiendo una gran cantidad de líneas de código de manera repetitiva aumentando considerablemente el tiempo invertido.

Por otro lado, los SP pueden también ejecutar diversas actividades, por ejemplo el cálculo del volumen de una esfera. Esto abre la línea de pensamiento, dejando claro que la creatividad y la habilidad del diseñador para programar los SP hace de éstos una gran herramienta.

El siguiente SP toma el diámetro de una circunferencia y calcula el volumen de la esfera que se puede formar con esa circunferencia.

```
CREATE OR REPLACE FUNCTION calcula_volumen_esfera (longitud_diametro double precision)
-- El parámetro de entrada de la función es la longitud del diametro de la circunferencia--
RETURNS float

--La función regresará un tipo de valor decimal--
AS
$$
BEGIN

--Esta función no cuenta con cuerpo en forma de consultas o procedimientos--
RETURN
((longitud_diametro/2)*(longitud_diametro/2)*(longitud_diametro/2)*3.14159265358979323846*4)/3;

--La salida que se requiere es calculada en la opción RETURN--
END;
$$
LANGUAGE 'plpgsql';
```

Figura 11.10 - SP de cálculo diverso.

5. Ejercicios.

(NOTA: Resuelve los siguientes ejercicios en relación al proyecto que realizarás durante el curso, en dado caso que no tengas un proyecto, utiliza la información en el apéndice NFL-ONEFA parte 11 al final de esta práctica para realizarlos)

1. Elabora un **SP** con dos o más parámetros de entrada y dos o más variables.
2. Elabora un **SP** que realice una o más consultas del estilo UPDATE o DELETE.

Entregables requeridos para prácticas subsecuentes:

- SPs implementados

6. Apéndice NFL-ONEFA parte 11

Resuelve las siguientes actividades utilizando procedimientos almacenados o a través de consultas utilizando funciones propias de SQL, puedes consultar el listado de supuestos del *Apéndice NFL-ONEFA parte 01* de *Especificación de Requerimientos* (Práctica 01) para recordar las fórmulas para calcular los valores solicitados:

1. Utilizando funciones aritméticas construye las siguientes consultas:
    a. Datos completos de los pateadores (punter) incluyendo su promedio de yardas por patada.
    b. Datos completos de los pateadores de lugar (kicker) incluyendo su porcentaje de efectividad.
    c. Datos completos de los receptores (receiver) incluyendo su promedio de yardas por recepción.
    d. Datos completos de los corredores (runner) incluyendo su promedio de yardas por acarreo.
    e. Datos completos de los mariscales de campo (quarterback) incluyendo su rating.
2. Utilizando funciones ya definidas en PostgreSQL construye una consulta que arroje los datos completos de los jugadores incluyendo su edad en años.
3. Construye un procedimiento almacenado que muestre los datos completos de los jugadores incluyendo la categoría a la que pertenecen de acuerdo a su edad.

```
-- Atributos derivados
-- Datos completos de los pateadores (punter) incluyendo su promedio de yardas por patada.
SELECT id_jugador, posicion, punts, net, longest, in20, (net/punts::FLOAT) AS avrg FROM punter;

-- Datos completos de los pateadores de lugar (kicker) incluyendo su porcentaje de efectividad.
SELECT id_jugador, posicion, fga, fgm, blocked, longest, (fgm/fga::NUMERIC(3,1))*100 AS pctg FROM kicker;

-- Datos completos de los receptores (receiver) incluyendo su promedio de yardas por recepción.
SELECT id_jugador, posicion, rec, yds, fumbles, tds, (yds/rec::FLOAT) AS avrg FROM receiver;

-- Datos completos de los corredores (runner) incluyendo su promedio de yardas por acarreo.
SELECT id_jugador, posicion, carries, fumbles, yds, tds, (yds/carries::FLOAT) AS avrg FROM runner;

-- Datos completos de los mariscales de campo (quarterback) incluyendo su rating.
SELECT id_jugador, posicion, tds, ints, yds, att, comp, ((8.4*yds+330*tds+100*comp-200*ints)/att::FLOAT) AS rating
FROM quarterback;

-- Datos completos de los jugadores incluyendo su edad en años.
SELECT id_jugador, apellido_jugador, nombre_jugador, fecha_nacimiento, universidad, AGE(fecha_nacimiento) AS edad
FROM jugador;
```

-- Procedimiento almacenado que muestre los datos completos de los jugadores
-- incluyendo la categoría a la que pertenecen de acuerdo a su edad.

```
CREATE OR REPLACE FUNCTION categoria()
 RETURNS TABLE (
        id_jugador INT,
        apellido_jugador VARCHAR(20),
        nombre_jugador VARCHAR(20),
        fecha_nacimiento DATE,
        universidad VARCHAR(30),
        categoria VARCHAR(30)
 )
 AS
$BODY$
DECLARE
   tupla jugador%ROWTYPE;
   anyos INTEGER;
BEGIN
FOR tupla IN
        SELECT *
        FROM jugador
LOOP
anyos :=  EXTRACT (years FROM age(tupla.fecha_nacimiento)) - 15; -- El -15 se agregó para reducir las edades
reales de los jugadores
IF ( anyos between 8 and 14 )
THEN
        RETURN NEXT;
        categoria := 'Infantil';
ELSEIF (anyos between 15 and 16 )
THEN
        RETURN NEXT;
        categoria := 'Juvenil';
ELSEIF (anyos between 17 and 18 )
THEN
        RETURN NEXT;
        categoria := 'Intermedia';
ELSEIF (anyos between 19 and 24 )
THEN
        RETURN NEXT;
        categoria := 'Mayor';
ELSE
        RETURN NEXT;
        categoria := 'Sin Categoría';
END IF;

id_jugador := tupla.id_jugador;
apellido_jugador := tupla.apellido_jugador;
nombre_jugador := tupla.nombre_jugador;
fecha_nacimiento := tupla.fecha_nacimiento;
universidad := tupla.universidad;
```

```
END LOOP;
RETURN NEXT;
END;
$BODY$ LANGUAGE plpgsql STABLE;

--SELECT * FROM categoria();
--SELECT * FROM categoria() WHERE id_jugador = 13;
```