Practica 10

Alumnos:
Castañon Maldonado Carlos Emilio
Chávez Zamora Mauro Emiliano
Gallegos Diego Cristian Ricardo
Navarro Santana Pablo César
Nepomuceno Escarcega Arizdelcy Lizbeth

Facultad de Ciencias
UNAM

Reporte - Práctica 10: Procedimientos y disparadores

Para esta práctica 10 incluimos la solución de cada uno de los códigos solicitados y la explicación de cómo funcionan.

1. Funciones (funcion.sql)

➢ Una función que reciba el identificador de veterinarios y regrese la edad del mismo.

```
CREATE OR REPLACE FUNCTION get_edad_veterinario(veterinarian_id INT)
RETURNS INT AS $$
BEGIN
    RETURN (
        SELECT EXTRACT(YEAR FROM AGE(CURRENT_DATE, fechNacimiento))
        FROM veterinario
        WHERE idPersona = veterinarian_id
    );
END; $$
LANGUAGE plpgsql;
```

Explicación: Recordando que como tal, no tenemos la edad de los veterinarios guardada y lo mas cercano que tenemos es su fecha de nacimiento, entonces procedemos a obtener la fecha de nacimiento del veterinario seleccionado para entonces usar `SELECT EXTRACT` de `YEAR FROM AGE` para de esa forma saber los años que han pasado desde esa fecha de nacimiento hasta el momento exacto de la consulta, dándonos de esa forma, una manera bastante precisa de obtener la edad a través de la fecha de nacimiento de un determinado veterinario.

➢ Una función que reciba el bioma y calcule el número de animales en ese bioma.

```
CREATE OR REPLACE FUNCTION get_numero_animales_en_Bioma(bioma_id INT)
RETURNS INT AS $$
BEGIN
    RETURN (
        SELECT COUNT(*)
        FROM jaula
        WHERE idBioma = bioma_id
    );
END; $$
LANGUAGE plpgsql;
```

Explicación: Recordando que en nuestro caso de uso, tenemos que cada bioma tiene a sus animales y todo animal tiene asignado solo una jaula (y cada jaula tiene exclusivamente un solo bioma y un solo animal asignado), tenemos que una forma de calcular el número de animales en un bioma se puede reducir en nuestro modelo a calcular cuantas jaulas pertenecen a un determinado bioma, con esto sabremos cuantos animales tiene un determinado bioma (esto considerando que no disponemos de jaulas vacías).

2. Procedimientos almacenados (SP.sql)

➢ Un SP el cual se encarga de registrar un cliente, en este SP, debes introducir la información del cliente y se debe encargar de insertar en la tabla correspondiente, es importante que no permitan la inserción de números o símbolos cuando sean campos relacionados a nombres.

```
CREATE OR REPLACE FUNCTION registrar_cliente(
    p_idPersona BIGINT,
    p_nombre VARCHAR(50),
    p_paterno VARCHAR(50),
    p_materno VARCHAR(50),
    p_genero CHAR(1)
) RETURNS VOID AS $$
BEGIN
    -- Verificar que los campos nombre, paterno y materno no contengan números o símbolos
    IF p_nombre ~ '[^a-zA-Z]' OR p_paterno ~ '[^a-zA-Z]' OR p_materno ~ '[^a-zA-Z]' THEN
        RAISE EXCEPTION 'Los campos nombre, paterno y materno no deben contener números o símbolos';
    END IF;

    -- Insertar los datos en la tabla cliente
    INSERT INTO cliente(idPersona, nombre, paterno, materno, genero)
    VALUES (p_idPersona, p_nombre, p_paterno, p_materno, p_genero);
END;
$$ LANGUAGE plpgsql;
```

Explicación: Como deseamos solo insertar clientes que cumplan con los parámetros establecidos, primero verificamos esto con el `if`, para que en caso de que nos pasen un nombre o apellido paterno/materno que no cumpla con la norma lo podamos rechazar y no insertar en cliente, en cambio, si nos pasan uno que si cumple con las normas, procedemos a insertarlo en cliente.

➢ Un SP que se encargue de eliminar un proveedor a través de su id, en este SP, se deberá eliminar todas las referencias del proveedor de las demás tablas.

```
CREATE OR REPLACE FUNCTION eliminar_proveedor(
    p_idPersona BIGINT
) RETURNS VOID AS $$
BEGIN
    --Verificar si el id dado, existe en la tabla proveedor
    if (SELECT NOT EXISTS (select p_idPersona from proveedor where idPersona = p_idPersona)) THEN
        RAISE EXCEPTION 'No existe el id persona en la tabla proveedor';
    END IF;

    -- Eliminar los datos del proveedor
    DELETE FROM proveedor where idPersona = p_idPersona;
    end;
    $$ language plpgsql;
```

Explicación: Primero obtenemos el id del proveedor que vamos a eliminar, después de eso verificamos que efectivamente exista ese id del proveedor, una vez hecho lo anterior borramos al proveedor, notese que no necesitamos de pasos adicionales ya que tenemos una politica de cascada en nuestro codigo.

3. Disparadores (Trigger.sql)

➢ Un trigger que se encargue de invertir el apellido paterno con el apellido materno de los proveedores.

```
CREATE OR REPLACE FUNCTION intercambia_apellidos()
RETURNS TRIGGER
as $$
DECLARE
    temporal_paterno VARCHAR(20);
    temporal_materno VARCHAR(20);
BEGIN
    SELECT paterno INTO temporal_paterno FROM proveedor WHERE new.idPersona = idPersona;
    SELECT materno INTO temporal_materno FROM proveedor WHERE new.idPersona = idPersona;
    UPDATE proveedor SET paterno = temporal_materno,materno = temporal_paterno
    WHERE NEW.idPersona = idPersona;
    RETURN null;
END;
$$ LANGUAGE plpgsql;

create TRIGGER cambia_apellidos
AFTER INSERT ON proveedor
FOR EACH ROW
EXECUTE PROCEDURE intercambia_apellidos();
```

Explicación: Este disparador ocupa una función auxiliar, porque según las clases impartidas por el profesor; Postgresql no admite definir directamente sobre el disparador (TRIGGER) lo que realizaremos. La función auxiliar *intercambia_apellidos* regresa un disparador (TRIGGER), peor al empezar ocupar un par de variables para extraer los apellidos que se le han asignado a una tupla de la tabla *Proveedor* al crearla, por eso mismo tienen que ser de tipo VARCHAR, asumimos que nadie tiene apellidos de más de 20 caracteres. Tenemos habilitada la facilidad de *new* puesto que existe sólo en inserciones o actualizaciones, así que aprovechamos el nuevo registro creado para extaer sus apellidos materno/paterno con uso de $SELECT$ y asignamos sus valores a nuestras variables. Después sólo debemos actualizar la tupla utilizando el identificador del registro en *new*, nos valemos de una instrucción $WHERE$ y como es un identificador, este cambio no se hace en ningún otro registro porque no pueden haber identificadores repetidos. Regresamos *null* puesto que aparte de lo que hicimos no deseamos que la función auxiliar nos ofrezca algún otro valor.

Al final sólo queda que nuestro disparador invoque a esa función sobre cada inserción en *proveedor* (por eso agregamos $FOR\ EACH\ ROW$), esto lo hace inmediatamente después de insertar la tupla, así que especificamos que es $AFERT\ INSERT\ ON$. Ejecuta el procedimiento que declaramos en la función *intercambia_apellidos* (el cual ya explicamos) y así consigue el intercambio de apellidos.

➢ Un trigger que se encargue de contar las personas que asisten a un evento, y agregarlo como atributo en evento. Cada vez que se inserte, se debera actualizar el campo.

```
create or replace function actualizarNoAsistentes()
returns trigger as $$
begin
    -- Verificamos si la columns noASistentes ya existe
    perform column_name
    from information_schema.columns
    where table_name = 'evento' and column_name = 'noasistentes';

    -- Sino existe, entonces agregamos la columna noAsistentes
    if not found then
        alter table evento add column noasistentes INTEGER default 0;
    end if;

    if TG_OP = 'INSERT' then
        update evento
```

```
        set noasistentes = (noasistentes + 1)
        where idevento = new.idevento;
    elsif TG_OP = 'DELETE' then
        update evento
        set noasistentes = (noasistentes - 1)
        where idevento = old.idevento;
    end if;
    return null;
end;
$$ language plpgsql;

-- TRIGGERS PARA LA ASISTENCIA DE EVENTOS
create trigger actualizarAsistentesCliente
after insert or delete on asistirCliente
for each row
execute function actualizarNoAsistentes();

create trigger actualizarAsistentesCuidador
after insert or delete on asistirCuidador
for each row
execute function actualizarNoAsistentes();

create trigger actualizarAsistentesProveedor
after insert or delete on asistirProveedor
for each row
execute function actualizarNoAsistentes();

create trigger actualizarAsistentesVeterinario
after insert or delete on asistirVeterinario
for each row
execute function actualizarNoAsistentes();
```

Explicación: Para poder ejecutarlo satisfactoriamente, antes de poblar la base de datos, debemos correr el código anterior, para poder llevar a cabo el conteo de numero de asistentes a un evento y después poblarla, ya que los registros viejos no los tomaría.
Dejando de lado eso, como no teníamos contemplado el número de asistentes en nuestro esquema, lo primero que se a hace es ver si esta la columna número de asistentes, en caso de que no la crea, y ya simplemente se auto incrementa el contador.
Notemos que tenemos una función y 4 triggers, eso se debe a que tenemos 4 tablas donde se guardan los registros de asistencias.