Universidad Nacional Autónoma de México
Facultad de Ciencias, 2023-I
Fundamentos de Bases de Datos

Práctica 9:(Opcional)
Procedimientos e disparadores.

Alumnos:

Galeana Vidaurri Rodrigo
Garcia Arce Marco Antonio
Pérez Romero Natalia Abigail
Rosales Jaimes Victor
Rojas Jarillo Mauricio

Profesor:

Gerardo Avilés Rosas

Ayudantes de teoría:

Gerardo Uriel Soto Miranda
Tania Naomi Barajas Pulido

Ayudantes de laboratorio:

Ricardo Badillo Macías
Carlos Augusto Escalona Navarro

1. Funciones

1. Función que reciba el identificador del empleado y regrese la edad de mismo

```sql
-- Una función que reciba el identificador de empleado y regrese la edad del mismo.
CREATE OR REPLACE FUNCTION obtenEdad(CHAR)
RETURNS INT
AS $$
DECLARE
    Edad INT;
    Empleado INT;
BEGIN
    empleado := (SELECT AnioNacimiento
                 FROM Gerente
                 WHERE RFCGerente=$1
                 UNION
                 SELECT AnioNacimiento
                 FROM Cuidador
                 WHERE RFCCuidador=$1
                 UNION
                 SELECT AnioNacimiento
                 FROM Vendedor
                 WHERE RFCVendedor=$1
                 UNION
                 SELECT AnioNacimiento
                 FROM Cajero
                 WHERE RFCCajero=$1
                 );
    --AnioActual
    Edad :=(SELECT EXTRACT(YEAR FROM (SELECT current_date)))-empleado;
    RETURN Edad;
END;
$$
LANGUAGE plpgsql;
```

Esta función declara las variables edad y empleado. En empleado almacena el resultado de la unión de las consultas del año de nacimiento de algún tipo (Gerente, Cuidador, Vendedor, Cajero) de empleado dado su RFC, el cual es el primer y único parámetro de la función. Luego en edad se extrae el año actual y se le resta el año de nacimiento de empleado. Y finalmente regresa edad.

2. Una función que reciba idVivero y regrese las ganancias de ese Vivero.

```sql
-- Una función que reciba idVivero y regrese las ganancias de ese Vivero.
-- Calcula la suma del monto de los pedido fisicos y en linea del vivero
CREATE OR REPLACE FUNCTION obtenGanancias(INT)
RETURNS INT
AS $$
DECLARE
    Sucursal INT;
    MontoPedidoLinea INT;
    MontoPedidoFisico INT;
    Total INT;
BEGIN
    Sucursal := $1;
    Total := 0;
    MontoPedidoLinea := (SELECT SUM(Monto) FROM (SELECT Monto
                                                 FROM PedidoLinea
                                                 WHERE NumSucursal=Sucursal)
                                                 AS PLinea);
    MontoPedidoFisico := (SELECT SUM(Monto) FROM (SELECT Monto
                                                  FROM PedidoFisico
                                                  WHERE NumSucursal=Sucursal)
                                                  AS PFisico);
    Total := Total+MontoPedidoLinea+MontoPedidoFisico;
    RETURN Total;
END;
$$
LANGUAGE plpgsql;
```

Para obtener las ganancias de un Vivero calcula la suma del monto de los pedido físicos y en linea del este. En la variable Sucursal almacena el numSucursal que se paso como parámetro y se utiliza para consultar el monto de los pedidos tanto en linea como fisicos del Vivero, a esta consulta se le aplica la suma, y se almacena en Total.

2. SP (Procedimiento almacenado)

1. 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

```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 PROCEDURE insertaCliente(id int, nom VARCHAR(50), aP VARCHAR(50), aM VARCHAR(50), Dir TEXT, DiaN int, MesN int, AnioN int)
AS $$
BEGIN
    IF nom IS NOT NULL AND (nom SIMILAR TO '[a-zA-Z]+') THEN
        IF aP IS NOT NULL AND (aP SIMILAR TO '[a-zA-Z]+') THEN
            IF aM IS NOT NULL AND (aM SIMILAR TO '[a-zA-Z]+') THEN
                INSERT INTO Cliente(idcliente,nombre,apellidop,apellidom,direccion,dianacimiento,mesnacimiento, anionacimiento) VALUES (id, nom , aP, aM, Dir, DiaN, MesN, AnioN);
            ELSE
                RAISE EXCEPTION 'Caracteres no permitidos' USING HINT ='Verifica que el apellido materno no contenga números o símbolos';
            END IF;
        ELSE
            RAISE EXCEPTION 'Caracteres no permitidos' USING HINT ='Verifica que el apellido paterno no contenga números o símbolos';
        END IF;
    ELSE
        RAISE EXCEPTION 'Caracteres no permitidos' USING HINT ='Verifica que el nombre no contenga números o símbolos';

    END IF;
END;
$$
LANGUAGE plpgsql;
```

Este método verifica que los campos relacionados con los nombres es decir: Nombre, ApellidoP y ApellidoM no contengan números o símbolos, es decir que sean cadenas de solo letras mayúsculas y minúsculas por eso se utiliza operador SIMILAR TO '[a-zA-Z]', además de no ser nulos. Y en caso de que el nombre que se intenta insertar no cumpla con las condiciones anteriores se indica en que campo se encontro el carácter invalido.

2. Un SP el cual se encarga de eliminar un cliente, en este SP debes introducir el identificador del cliente y cuando elimines ese cliente se debe eliminar todas sus referencias.

```sql
--Un SP el cual se encarga de eliminar un cliente, en este SP debes introducir el identificador del cliente y cuando elimines ese cliente se debe eliminar todas sus referencias.

CREATE OR REPLACE PROCEDURE eliminarCliente(id int)
AS $$
BEGIN
DELETE FROM CorreoCliente WHERE IdCliente=id;
DELETE FROM TelefonoCliente WHERE IdCliente=id;
DELETE FROM PedidoLinea WHERE IdCliente=id;
DELETE FROM PedidoFisico WHERE IdCliente=id;
DELETE FROM Cliente WHERE IdCliente=id;
END;
$$
LANGUAGE plpgsql;
```

La declaración DELETE de PostgreSQL permite eliminar una o más filas de una tabla, donde se satisface cierta condición, en este caso: elimina las filas donde el IdCLiente de las tablas CorreoCliente, TelefonoCliente,PedidoLinea,PedidoFisico,Cliente corresponde al id del cliente que se paso como parametro.

3. Triggers

1. Un trigger que se encargue de invertir el apellido paterno con el apellido materno de los empleados.

```sql
-- Un trigger que se encargue de invertir el apellido paterno con el apellido materno de los empleados.
CREATE OR REPLACE FUNCTION invertirApellidos() RETURNS TRIGGER
AS
$$
DECLARE
    aPaterno VARCHAR(50);
    aMaterno VARCHAR(50);
BEGIN
    aPaterno := (SELECT ApellidoP FROM TG_TABLE_NAME);
    aMaterno := (SELECT ApellidoM FROM TG_TABLE_NAME);
    IF TG_OP = 'UPDATE' THEN
        UPDATE TG_TABLE_NAME SET ApellidoP=aMaterno, ApellidoM=aPaterno;
    END IF;
    RETURN NULL;
END;
$$
LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS invertirGerente ON Gerente;
CREATE TRIGGER invertirGerente
    AFTER UPDATE
    ON Gerente
    FOR EACH ROW
    EXECUTE PROCEDURE invertirApellidos();


DROP TRIGGER IF EXISTS invertirCuidador ON Cuidador;
CREATE TRIGGER invertirCuidador
    AFTER UPDATE
    ON Cuidador
    FOR EACH ROW
    EXECUTE FUNCTION invertirApellidos();


DROP TRIGGER IF EXISTS invertirVendedor ON Vendedor;
CREATE TRIGGER invertirVendedor
    AFTER UPDATE
    ON Vendedor
    FOR EACH ROW
    EXECUTE PROCEDURE invertirApellidos();


DROP TRIGGER IF EXISTS invertirCajero ON Cajero;
CREATE TRIGGER invertirCajero
    AFTER UPDATE
    ON Cajero
    FOR EACH ROW
    EXECUTE PROCEDURE invertirApellidos();
```

Dado que en nuestro diseño un Empleado puede ser Gerente, Cuidador, Vendedor o Cajero es necesario un trigger para cada una da las tablas, sin embargo es posible usar la misma función: `invertirApellidos` para todos, aprovechando la variable `TG_TABLE_NAME` que almacena la tabla sobre la que el trigger actuará. El trigger se ejecutara después de la instrucción `UPDATE` y sobre todas la filas de la tabla del tipo de empleado aplicara la función `invertirApellidos`. Esta función utiliza variables temporales para almacenar los apellidos y en cuanto se realiza UPDATE en una de las tablas de los empleados invertir los apellidos.

2. Un trigger que se encargue de evitar que se pueda modificar y borrar de la tabla plantas.

```sql
-- Un trigger que se encargue de evitar que se pueda modificar y borrar de la tabla plantas.
CREATE OR REPLACE FUNCTION evitarMod()RETURNS TRIGGER
AS
$$
BEGIN
    IF TG_OP = 'UPDATE' THEN
        RAISE EXCEPTION 'No permitido' USING HINT ='No es posible modificar la tabla plantas';
    END IF;
    IF TG_OP = 'DELETE' THEN
        RAISE EXCEPTION 'No permitido' USING HINT ='No es posible borrar en la tabla plantas';
    END IF;
    IF TG_OP = 'INSERT' THEN
        RAISE EXCEPTION 'No permitido' USING HINT ='No es posible insertar en la tabla plantas';
    END IF;
    RETURN NULL;
END;
$$
LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS modPlantaA ON PlantaAfricana;

CREATE TRIGGER modPlantaA
    AFTER UPDATE OR DELETE OR INSERT
    ON PlantaAfricana
    FOR EACH ROW
    EXECUTE PROCEDURE evitarMod();

DROP TRIGGER IF EXISTS modPlantaC ON PlantaCactus;
CREATE TRIGGER modPlantaC
    AFTER UPDATE OR DELETE OR INSERT
    ON PlantaCactus
    FOR EACH ROW
    EXECUTE PROCEDURE evitarMod();
```

De manera similar, en nuestro diseño las plantas pueden ser PlantaAfricana o PlantaCactus así que es necesario un trigger para cada tabla pero puede disparar la función `evitarMod()` la cual despues de cualquier llamada a una instrucción que modifique la tabla, es decir `UPDATE`, `DELETE`, `INSERT`, lanza una excepción que indica que la instrucción no está permitida.