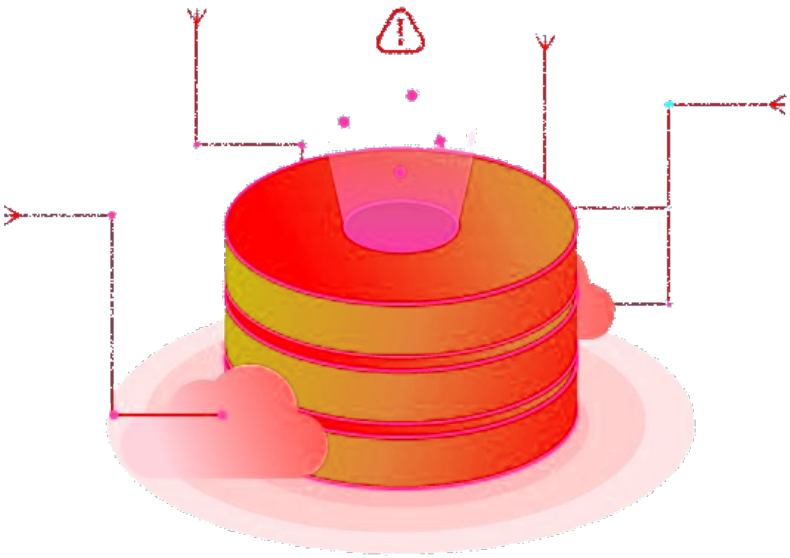
Universidad Nacional Autónoma De México
Facultad De Ciencias, 2024-I
Fundamentos De Bases De Datos



Práctica 10:

Procedimientos e disparadores

Profesor:
Gerardo Avilés Rosas

Ayudantes de teoría:
Gerardo Uriel Soto Miranda
Valeria Fernanda Manjarrez Angeles

Ayudantes de laboratorio:
Ricardo Badillo Macías
Jerónimo Almeida Rodríguez

Procedimientos Almacenados (Stored Procedures):

Los procedimientos almacenados son bloques de código que permiten agrupar y organizar sentencias SQL, que se ejecutan al invocarlos.

La estructura de un procedimiento consta de una cabecera, una sección de declaración de variables y el bloque begin ... end que encierra las acciones a realizar.

Funciones:

En **PostgreSQL** encontramos **funciones** (con las mismas caracteristicas de los **Procedimientos Almacenados**), solo que las funciones contienen ademas la cláusula return.

Una función acepta parámetros, se invoca con su nombre y retorna un valor. Para crear una función empleamos la instrucción **CREATE FUNCTION** o **CREATE OR REPLACE FUNCTION**. Si empleamos *OR REPLACE*, se sobreescribe (reemplaza) una función existente; si se omite y exite una función con el nombre que le asignamos, saldra un mensaje de error.

Uno de los lenguajes soportados para programar funciones en **PostgreSQL** es el **PL/pgsql**, que se desarrolló para este SABD y es parecido a **PL/SQL** de **Oracle**.
Una vez conectado a nuestra base de datos para poder utilizar pgsql, es necesario la siguiente instrucción:

```
CREATE LANGUAGE plpgsql;
```

En la instalación de **WINDOWS** ya se incluye este paquete, en distribucciones linux es necesario especificarlo.

Estructura basica de las funciones

```
CREATE OR REPLACE FUNCTION nombreFuncion(param,param,param,...)
    RETURNS tipoDeDatoDeRetorno
    AS $$
    DECLARE
        variable;
        variable;
    BEGIN
        sentencia; --Comentario
        sentencia; /*Bloque de Comentarios*/
        sentencia;
        RETURN retorno;
    END;
    $$
    Language plpgsql;
```

Invocación de Funciones

- Como columna de un **SELECT**.
- Condiciones en cláusulas **WHERE y HAVING.**
- Cláusulas **ORDER BY y GROUP BY**.
- Cláusula **VALUES** de un comando **INSERT**.
- Cláusula **SET** de un comando **UPDATE**.

Restricciones en Funciones

- No se permiten comandos **INSERT, UPDATE o DELETE**, dentro de la implementación de una función.
- La función no puede llamar a otro subrutina que rompa una de las restricciones anteriomente indicadas.

Eliminación de Funciones

Una vez que se haya creado la función, quizas sea necesario elminarla de la base de datos, para estas situaciones tenemos la siguiente sintaxis:

```
DROP FUNCTION nombreFuncion;
```

Stored Procedures(Procedimientos Almacenados)

Creación de SP

Al crear un procedimiento almacenado, las instrucciones que contiene se analizan para verificar si son correctas sintácticamente. Si se encuentra algún error, el procedimiento se complila, pero aparece un mensaje con advertencias que indica tal situación.

Los procedimientos almacenados pueden hacer referencia a tablas, vistas, a funciones definidas por el usuario, a otros procedimientos almacenados.

Un procedimiento almacenado pueden incluir cualquier cantidad y tipo de instrucciones DML **(INSERT, UPDATE, DELETE)**, no instrucciones DDL **(CREATE, DROP, ALTER)**.

Para crear un procedimiento almacenado empleamos la instrucción **CREATE PROCEDURE para versiones a partir de postgres 11**, para **versiones anteriores a esta se usa el CREATE FUNCTION.**

Estructura basica de los procedimientos:

```
CREATE OR REPLACE PROCEDURE nombreProcedimiento
AS $$
BEGIN
   instrucciones
END;
$$
Language plpgsql;
```

Invocación de SP

Para ejecutar el procedimiento utilizamos la cláusula execute:

```
CALL nombreProcedimiento(parametros)

EXECUTE nombreProcedimiento(parametros)
```

Eliminación de SP

Los procedimientos almacenados se eliminan con **DROP PROCEDURE**.

```
DROP PROCEDURE nombreProcedimiento;
```

Parametros en SP.

Los procedimientos almacenados pueden recibir y devolver información, para ello se emplean parámetros. Los **parametros de entrada** posibilitan pasar infromación a un procedimiento. Para que un **procedimiento almacenado admita parámetros de entrada se deben declarar al crearlo**.

Los **parámetros de salida** no pasan información a un procedimiento, estos solo son ocupada para que el procedimientos **guarden algún valor en este parámetro** y al final de su ejecución regrese este parámetro con los valores

que se definan dentro del SP. **Para que un procedimiento almacenado admita parámetros de salida se deben declarar al crearlo**.

Los **parámetros de entrada-salida** pueden pasar información a procedimientos, estos son ocupadas para que el procedimiento guarde algún valor en este parámetro, y al final de su ejecución regresa este parámetro con los valores que se definan dentro del SP y al mismo tiempo puede ocupar el valor de entrada de este parámetro para tomar alguna decisión durante la ejecucón del SP. **Para que un procedimiento almacenado admita parámetros de entrada-salida se deben declarar al crearlo**.

```
CREATE OR REPLACE PROCEDURE nombreProcedimiento(parametro IN tipoDato) --Parametro entrada
    AS $$
    BEGIN
        instrucciones;
    END;
    $$
    Language plpgsql;

CREATE OR REPLACE PROCEDURE nombreProcedimiento (parametro OUT tipoDato) --Parametro salida
    AS $$
    BEGIN
        instrucciones;
    END;
    $$
    Language plpgsql;

CREATE OR REPLACE PROCEDURE nombreProcedimiento (parametro IN OUT tipoDato) --Parametro
     entrada-salida
    AS $$
    BEGIN
        instrucciones;
    end;
    $$
    Language plpgsql;
```

Creación de variables en SP

Los procedimientos almacenados pueden contener en su definición, variables locales, que existen durante el procedimiento.
La sintaxis para declarar variables dentro de un procedimiento almacenado es:

```
CREATE OR REPLACE PROCEDURE nombreProcedimiento (parametro IN tipoDato)
    AS $$
        nombreVariable tipo;
    BEGIN
        instrucciones;
    END;
    $$
    Language plpgsql;
```

Las variables se definen antes del bloque de sentencias; pueden declararse varias.

Disparadores (Trigger):

Disparadores de inserción a nivel de fila (insert trigger for each row

Un trigger para el evento de inserción a nivel de sentencia, es decir, se dispara una vez por cada sentencia INSERT sobre la tabla asociada. En caso de que una sola sentencia insert ingrese varios resgistros en la tabla asociada, el trigger se disparará una sola vez, si queremos que se active una vez por cada registro afectado, debemos indicarlo con for each row. La siguiente es la sintaxis para crear un trigger de inserción a nivel de fila, se dispare una vez por cada fila ingresada sobre la tabla especificada:

```
CREATE OR REPLACE TRIGGER nombreDisparador
    [AFTER|BEFORE] INSERT
    ON nombreTabla
    FOR EACH ROW

    BEGIN
        CUERPODelTrigger;
    END nombreDisparador;
```

Trigger de múltiples eventos

Un trigger puede definirse sobre más de un evento, en tal caso se separan con or. Sintaxis:

```
CREATE OR REPLACE TRIGGER nombreDisparador
    [AFTER|BEFORE]
    OF campo -- Si alguno de los eventos es UPDATE
    [INSERT|UPDATE|DELETE]
    ON nombreTabla
    FOR EACH [ROW|STATEMENT] -- puede ser a nivel de sentencia o de fila
    BEGIN
        CUERPODelTrigger; --sentencias
    END nombreDisparador;
```

Si el trigger se define para más de un evento desencadenante, en el cuerpo del mismo se puede emplear un condicional para controlar cuál operación disparó el trigger. Esto permite ejecutar bloques de código segun la clase de acción que disparó el desencadenador.

Para identificar el tipo de operación que disparó el trigger empleamos inserting, updating y deleting.
Ejemplo:
El siguiente trigger está definido a nivel de sentencia, para los eventos insert, update y delete, cuando se modifican los datos de A, se almacena en la tabla B el nombre del usuario, la fecha y el tipo de modificación que alteró la tabla:

- Si se realizó una inserción, se almacena la inserción.
- Si se realizó una actualización(update), se almacena, la actualización.
- Si se realizó una eliminación (delete) se almacena borrado.

```
CREATE OR REPLACE FUNCTION check_cambios_a()RETURNS TRIGGER
AS
$$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO b(nombreUsuario,fecha,tipo) VALUES (user,current_date,'inserción');
    END IF;
    IF TG_OP = 'UPDATE' THEN
        INSERT INTO b(nombreUsuario,fecha,tipo) VALUES(user,current_date,'actualizacion');
    END IF;
    IF TG_OP = 'DELETE' THEN
        INSERT INTO b(nombreUsuario,fecha,tipo) VALUES(user,current_date,'borrado');
    END IF;
    RETURN NULL;
END;
$$
LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER registra_cambios_a
    AFTER INSERT OR UPDATE OR DELETE
    ON A
    FOR EACH ROW
    EXECUTE PROCEDURE check_cambios_a();
```

Si ejecutamos un INSERT sobre A, el disparador se activa entrando por el primer if; si ejecutamos un UPDATE el trigger se dispara entrando por el segundo IF; si ejecutamos un DELETE el desencadenador se dispara entrando por el tercer IF.

Disparador (old y new)

Cuando trabajamos con trigger a nivel de fila, Oracle provee de dos tablas temporales a las cuales se pueden acceder, que contienen los antiguos y nuevos valores de los campos del registro afectado por la sentencia que disparó el trigger. El nuevo valor es :NEW y el viejo valor es :OLD. Para referirnos a ellos debemos especificar su campo separado por un punto :NEW.campo y :OLD.campo.

El acceso a estos campos depende del evento del disparador. En un trigger disparador por un insert, se puede acceder al campo :NEW unicamente, el campo :OLD contiene null. En una inserción se puede emplear :NEW para escribir nuevos valores en las columnas de la tabla.

En un trigger que se dispara con UPDATE, se puede acceder a ambos campos. En una actualización se pueden comparar los valores de :NEW y :OLD.

En un trigger de borrado, unicamente se puede acceder al campo :OLD ya que el campo :NEW no existe luego que el registro es eliminado, el campo :NEW contiene null y no puede modificarse.
Los valores de :OLD y :NEW están disponibles en triggers AFTER y BEFORE.

El valor de :NEW puede modificarse en un trigger BEFORE, es decir, se puede acceder a los nuevos valores antes que se ingresen en la tabla y cambiar los valores asignados a :NEW.campo por otro valor.

El valor de :NEW no puede modificarse en un trigger AFTER, esto es porque el trigger se activa luego que los valores de :NEW se alamacenaron en la tabla. El campo :OLD nunca se modifica, sólo puede leerse. En el cuerpo el trigger, los campos :OLD y :NEW deben estar precedidos por ":"(dos puntos), pero si está en WHEN no.

Disparador de actualización - campos (UPDATING)

En un trigger de actualización a nivel de fila, se puede especificar el nombre de un campo en la condición updating para determinar cuál campo ha sido actualizado.
Sintaxis:

```
CREATE OR REPLACE TRIGGER nombreDisparador
[AFTER|BEFORE] UPDATE ...
BEGIN
    IF UPDATING ('campo') then
    ...
    END IF;
END nombreDisparador;
```

Disparadores (Habilitar y Deshabilitar)

Un disparador puede estar en dos estaods: habilidato (ENABLE) o deshabilitado (DISABLE). Cuando se crea un trigger por defecto está habilitado. Se puede deshabilitar un trigger para que no se ejecute. Un trigger deshabilitado sigue existiendo, pero al ejecutar una instrucción que lo dispara, no se activa.
Sintaxis:

```
ALTER TRIGGER nombreDisparador DISABLE;

ALTER TRIGGER nombreDisparador ENABLE;
```

Se pueden habilitar o deshabilitar todos los trigger establecidos sobre una tabla especifica, se emplea la siguiente sentencia.

```
ALTER TABLE nombreTabla DISABLE ALL TRIGGERS; --Deshabilita
ALTER TABLE nombreTabla ENABLE ALL TRIGGERS; -- Habilita
```

Podemos saber si un trigger está o no habilitado consultando el diccionario user_triggers, en la columna status aparece ENABLE si esta habilitado y DISABLE si esta deshabilitado.

Eliminación de Disparadores

Para eliminar un trigger se emplea la siguiente sentencia:

```
DROP TRIGGER nombreDisparador;
```

Si se llega eliminar una tabla, se eliminara tambien todos los triggers establecidos sobre ella.

Actividades.

Utilizando la base de datos desarrollada a lo largo del semestre, se debe realizar lo siguiente:

- Deberan hacer un script llamado funcion.sql donde creen la siguiente funcion.
  i. Una función que reciba el identificador de veterinarios y regrese la edad del mismo.
  ii. Una función que reciba el bioma y calcule el número de animales en ese bioma.
- Deberan hacer un script llamado SP.sql donde realices el siguiente procedimiento:
  i. 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.
  ii. Un SP que se encargue de eliminar un proveedor a traves de su id, en este SP, se debera eliminar todas las referencias del proveedor de las demas tablas.
- Deberan hacer un script llamado Trigger.sql donde hagan el siguiente disparador:
  i. Un trigger que se encargue de invertir el apellido paterno con el apellido materno de los proveedores.
  ii. 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.
- En el pdf de la práctica deberan explicar cada función, procedimiento almacenado y trigger que realizaron.

Figura 1: *Actividades.*

Entregables.

Deberán subir un archivo con formato *zip* a *Google Classroom*, de acuerdo a lo indicado en los lineamientos de entrega. Debe de estar organizado de la siguiente manera, (suponiendo que el nombre del equipo que está entregando es *Dream Team*).

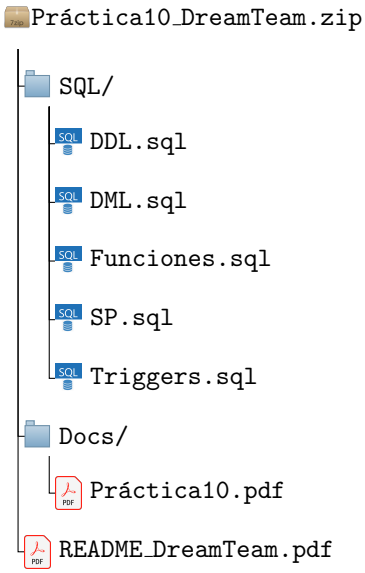


Figura 2: *Entregables.*

Nota.

Para cualquier duda o comentario que pudiera surgirles al hacer este trabajo, recuerden que cuentan con la asignación de este entregable en el grupo de *Classroom*, en donde seguramente encontrarás las respuestas que necesites.

Figura 3: *Nota.*