Universidad Nacional Autónoma de México

Facultad de Ciencias

Fundamentos de Bases de Datos

Práctica 10. Programación de las Bases de Datos

Equipo : eSosQLones

Estrada Garcia Luis Gerardo	319013832
Jiménez Hernández Allan	420003478
Mancera Quiroz Javier Alejandro	319274831
Mora Hernández Dulce Julieta	319236448
Peña Nuñez Axel Yael	318279754

Actividades

- En el pdf de la práctica deberan explicar cada función, procedimiento almacenado y trigger que realizaron.
  Funciones
  - *calcular_edad_veterinario*

```
CREATE OR REPLACE FUNCTION calcular_edad
_veterinario(id_veterinario INT) RETURNS INT AS $$
DECLARE
    fecha_nacimiento DATE;
BEGIN
    -- Encuentra la fecha de nacimiento del
    -- veterinario a partir del id_veterinario
    SELECT p.fecha_nac INTO fecha_nacimiento
    FROM persona p
    JOIN trabajador t ON p.id_persona = t.id_persona
    JOIN veterinario v ON t.rfc = v.rfc
    WHERE p.id_persona = id_veterinario;

    -- Calcula la edad del veterinario bas ndose
    -- en la fecha actual y la fecha de nacimiento
    RETURN EXTRACT(YEAR FROM age(CURRENT_DATE, fecha_nacimiento));
END;
```

`Explicación`
La función recibe el identificador de veterinarios y regrese la edad del mismo.

La función recibe un `INT` que indica el id del veterinario y realiza lo siguiente

Primero se declara una variable llamada 'fecha_nacimiento' que es de tipo DATE, y por medio de una consulta obtiene la fecha de nacimiento del veterinario para después almacenarla en la variable que habiamos creado.

La consulta realiza un join entre las tablas de persona y trabajador utilizando la columna id y a la tabla obtenida, le realiza otro join más utilizando la columna rfc. De la tabla obtenida toma la fechas de nacimiento de la persona que coincide con el id proporcionado y almacena la fecha de nacimiento en la variable 'fecha_nacimiento'.

Después con ayuda de la función *age*, toma la fecha actual y la fecha de nacimiento para calcular la diferencia. Después con ayuda de *extract* extrae el año de la diferencia que obtuvimos anteriormente.

Finalmente, devuelve un INT que contiene la edad del veterinario

- *calcular_numero_animales_bioma*

```
CREATE OR REPLACE FUNCTION calcular_numero_animales
_bioma(tipo_bioma VARCHAR) RETURNS INT AS $$
DECLARE
    numero_animales INT;
BEGIN
    -- Obtiene el numero de animales del
    -- bioma especificado por el tipo
    SELECT num_anim INTO numero_animales
    FROM bioma
    WHERE tipo = tipo_bioma;

    RETURN numero_animales;
END;
```

`Explicación`
La función recibe el bioma y calcula el número de animales en ese bioma.

La función recibe un `VARCHAR` que indica el tipo de bioma al que le queremos calcular el número de animales con los que cuenta.

Primero se declara una variable llamada 'numero_animales' que es de tipo INT. Después realiza una consulta de la cual se obtiene el número de animales(y se guarda en numero_animales) que pertenecen al tipo de bioma recibido en la función.

Finalmente, la función devuelve el valor que contiene 'numero_animales'

Procedimientos almacenados

- *registrar_cliente*

```
create or replace procedure registrar_cliente(id_persona int,
                    nombre varchar, apellidopat varchar,
                    apellidomat varchar,
                    genero char, fecha_nac date,
```

```
                    fechain_con date, fechafin_con date,
                    estado varchar, num_int int, num_ext int,
                    colonia varchar, calle varchar,
                    id_cliente int)
as $$
begin
        if(nombre ~* '[^A-Za-z]' or apellidopat ~* '[^A-Za-z]'
            or apellidomat ~* '[^A-Za-z]') then
                raise exception 'Algun apellido o nombre contiene
-----caracteres especiales o numeros';
        else
            insert into persona values (id_persona, nombre,
        apellidopat, apellidomat, genero, fecha_nac,
        fechain_con, fechafin_con, estado, num_int, num_ext,
        colonia, calle);
        insert into cliente values (id_cliente, id_persona);
        end if;
end;
$$
language plpgsql;
```

Explicación

El procedimiento recibe todos los parámetros necesarios para poder insertar a una persona, ya que sin esto no se puede generar un nuevo cliente, y el id_cliente que tendrá el cliente a insertar.

Primero se pasa por un if que verifica si el nombre, el apellido materno o el apellido paterno contienen algún caracter que no sea alfabético.

Si se cumple la condición manda una excepción diciendo que algún nombre o apellido contiene caracteres especiales o numeros.

Después si no se cumple la condición, entonces, inserta los datos requeridos en persona para crear a la persona y así poder luego crear al cliente. Si hay un error en los datos a ingresar será verificado por el archivo ddl.sql.

Finalmente, en el archivo SP.sql se dejan algunas instrucciones extra para probar el procedimiento.

- *eliminar_proveedor*

```
create or replace procedure eliminar_proveedor
```

```
(id_proveedor varchar)
as $$
begin
        delete from proveedor where rfc = id_proveedor;
        delete from trabajador where rfc = id_proveedor;
end;
$$
language plpgsql;
```

`Explicación`
El procedimientp recibe un `VARCHAR` que indica el id del proveedor, que es su rfc, y realiza lo siguiente

Primero elimina de la tabla proveedor al proveedor con el id dado, si ninguno coincide no hará nada.

Si existe el proveedor a eliminar, entonces, también lo elimina de trabajador, ya que es donde está referenciado igualmente la tabla proveedor.

No termina de eliminar de persona al proveedor con el id_persona asociado en trabajador para ese proveedor, puesto que, puede que ese proveedor sea un cliente, así que solo se elimina de las tablas donde esa persona está asociada como proveedor con su rfc y de ninguna otra tabla donde no se cumpla esto no se eliminará.

Finalmente, en el archivo SP.sql se dejan algunas instrucciones extra para probar el procedimiento.

Triggers

- Trigger que se encarga de invertir el apellido paterno con el apellido materno de los proveedores.
  `Explicación`

  1. Primero creamos una tabla donde se guardaran los registros, dicha tabla sera llamada 'persona1' y contara con los mismos atributos que cuenta la tabla 'persona'
  2. Después creamos el disparador.
     Primero se crea la función *invierte*, en ella declaramos todas las variables que de igual forma están declaradas en la tabla 'persona1'

     ```
     create or replace function invierte()
     ```

```
returns trigger
as
$$
declare
        idN int;
        nombreN text;
        apellidopatN varchar(50);
        apellidomatN varchar(50);
        generoN char;
        fecha_nacN date;
        fechain_conN date;
        fechafin_conN date;
        estadoN varchar(50);
    num_intN int;
        num_extN int;
        coloniaN text;
        calleN text;
```

Después, verificamos si la operación a realizar es una inserción, ya que requerimos una para poder invertir los datos

```
    if TG_OP = 'INSERT' then
```

En caso de que lo sea, obtenemos los datos del proveedor

```
idN := (select id_persona from trabajador t join proveedor p
on t.rfc = p.rfc where new.rfc = p.rfc);
nombreN := (select nombre from trabajador t join persona p
on t.id_persona = p.id_persona where idN = p.id_persona);
apellidopatN := (select apellidopat from trabajador t join
persona p on t.id_persona = p.id_persona where idN =
p.id_persona);
apellidomatN := (select apellidomat from trabajador t join
persona p on t.id_persona = p.id_persona where idN =
p.id_persona);
generoN := (select genero from trabajador t join persona p
on t.id_persona = p.id_persona where idN = p.id_persona);
fecha_nacN := (select fecha_nac from trabajador t join
persona p on t.id_persona = p.id_persona where idN =
p.id_persona);
fechain_conN := (select fechain_con from trabajador t join
persona p on t.id_persona = p.id_persona where idN =
p.id_persona);
fechafin_conN := (select fechafin_con from trabajador t join
persona p on t.id_persona = p.id_persona where idN =
p.id_persona);
```

```
estadoN := (select estado from trabajador t join persona p
on t.id_persona = p.id_persona where idN = p.id_persona);
num_intN := (select num_int from trabajador t join persona p
on t.id_persona = p.id_persona where idN = p.id_persona);
num_extN := (select num_ext from trabajador t join persona p
on t.id_persona = p.id_persona where idN = p.id_persona);
coloniaN := (select colonia from trabajador t join persona p
on t.id_persona = p.id_persona where idN = p.id_persona);
calleN := (select calle from trabajador t join persona p
on t.id_persona = p.id_persona where idN = p.id_persona);
insert into persona1 values(idN,nombreN,apellidomatN,
apellidopatN,generoN,fecha_nacN,fechain_conN,
fechafin_conN,estadoN,num_intN,num_extN,coloniaN,
calleN);
raise notice 'Se agrego al proveedor: %', nombreN;
```

y en caso contrario, lanzamos una excepción

```
raise exception 'No se inviertieron los apellidos, se
invierten cuando se agrega un registro en la tabla proveedor';
```

y finalmente se devuelve NULL ya que la función es un disparador

```
return null;
```

3. Después, habilitamos nuestro disparador

```
create or replace trigger check_invierte
after insert on proveedor
for each row
execute function invierte();
```

El codigo anterior permite que cada que realicemos una inserción a proovedor, se ejecute el disparador (after insert on proveedor)

Después, *for each row* indica que el disparador ejecutara para cada fila de la que realizamos inserción, realizamos la aplicación de *invierte()*, es decir, se invertiran los apellidos y se agregaran a la tabla persona1.

– Trigger que se encarga de contar las personas que asisten a un evento, y lo agregar como atributo en evento. Cada vez que se inserta, se actualiza el campo.
`Explicación`

1. Primero crearemos una tabla llamada 'evento1' que tendrá los mismos atributos que la tabla 'evento'

2. Lo siguiente que realizaremos será crear el disparador como sigue
Creamos la función *contar_asistentes*, primero verificamos si la operación a realizar es una inserción incializamos las variables como sigue

```
if TG_OP = 'INSERT' then
        cont := 0;
        iter := 0;
        tam := pg_column_size(new.listaasistentes);
        car := substring(new.listaasistentes from 1 for 1);
```

y con ayuda de un ciclo while contamos la cantidad de asistentes que hay en la lista

```
while (iter < tam)
        loop
        if (car = ';') then
                cont := cont + 1;
        end if;
         iter := iter + 1;
                car := substring(new.listaasistentes
            from iter for 1);
        end loop;
                cont := cont + 1;
                insert into evento1 values(new.id_evento,
        new.listaasistentes,new.capacidad,
        new.tipoevento,new.fecha,cont);
                raise notice 'Se agrego al evento con id: %',
        new.id_evento;
                return null;
```

Cuando se termina de ejecutar el ciclo while, se inserta la información en la tabla que generamos previamente, 'evento1' y devolvemos null

Si no entra al primer if, verificamos si la operación a realizar es una eliminación, si lo es, entonces eliminamos el evento con el id que nos proporcionan de la tabla 'evento1' y notificamos que se realizo la eliminación.

```
        if TG_OP = 'DELETE' then
                delete from evento1 where id_evento =
        old.id_evento;
                raise notice 'Se elimino al evento con id: %',
        old.id_evento;
        end if;
```

Si no entramos a ninguno de los dos if's pasados, la operación a realizar es una actualización, la actualización primero cuenta el numero

de asistentes que hay en la lista por medio del ciclo while, una vez terminado dicho loop, elimina la información para insertar nueva en la tabla 'evento1', finalmente, notifica que se realizo la actualización.

```
if TG_OP = 'UPDATE' then
        cont := 0;
        iter := 0;
        tam := pg_column_size(new.listaasistentes);
        car := substring(new.listaasistentes from 1 for 1);
        while (iter < tam)
                loop
                if (car = ';') then
                        cont := cont + 1;
                        end if;
   iter := iter + 1;
        car := substring(new.listaasistentes from iter for 1);
                end loop;
        cont := cont + 1;
        delete from evento1 where id_evento = old.id_evento;
        insert into evento1 values(old.id_evento,
    new.listaasistentes,new.capacidad,
    new.tipoevento,new.fecha,cont);
        raise notice 'Se actualizo el evento con id: %',
    old.id_evento;
        end if;
        return null;
```

Como se trata de un disparador, devolvemos NULL

```
return null;
```

3. Después, habilitamos nuestro disparador

```
create or replace trigger check_contar_asistentes
after insert or update or delete on evento
for each row
execute function contar_asistentes();
```

Al habilitar el disparador, podemos realizar ya sea, una inserción, una eliminación o una actualización a evento, al ejecutar el disparador (after insert on evento)

Después, *for each row* indica que el disparador ejecutara para cada fila de la que realizamos inserción, eliminación o actualización, realizaremos la aplicación de nuestro disparador.