PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

TALLER DE BASES DE DATOS

JEAN RODRÍGUEZ

INFORME DE TALLER VIII

ENERO, 2020

Índice

Lista de Figuras

1. Introducción

Los modelos de negocio actuales manejan cantidades enormes de datos de los clientes en cada empresa, sea grande o pequeña, lo que evidencia una clara necesidad de un software capaz de procesar las solicitudes requeridas. Cuando una entidad desea minimizar la complejidad de sus operaciones y mantener tiempos de respuesta aceptables, se debe dejar en claro un diseño simple y eficiente que sea mantenible y facilite la realización de cambios a futuro.

Con la intención de demostrar un correcto funcionamiento en la base de datos propuesta, se deja a disposición una serie de consultas para el motor PostgreSql que entregan respuestas de manera rápida y efectiva, utilizando como base el script inicial para cargar la base de datos, y su correspondiente archivo de valores separado por comas. Además, se deja a disposición una copia de la base de datos posterior a las modificaciones realizadas luego de realizar los ejercicios propuestos, que se encuentra en formato SQL plano para cargar mediante una query.

2. Resolución de ejercicios

A continuación se presenta el conjunto de preguntas y respuestas correspondientes en el lenguaje PostgreSQL, que se enfocan en resolver la necesidad de manejar eventos en ciertas circunstancias dadas denominadas disparadores.

1.-Construya un trigger llamado Artículos para postgresql que garantice que, al insertar un nuevo registro en la tabla Artículos, se guarda la fecha y hora de la inserción en el campo Auditoría.

Para crear un disparador en PostgreSQL, primero debemos definir un procedimiento almacenado y luego ejecutarlo desde un disparador que haga uso de la tabla (o query) que contenga los datos a manejar. En este ejercicio, un pseudo-código (es decir, sin trabajar en PostgreSQL) básico del procedimiento seria de la forma:

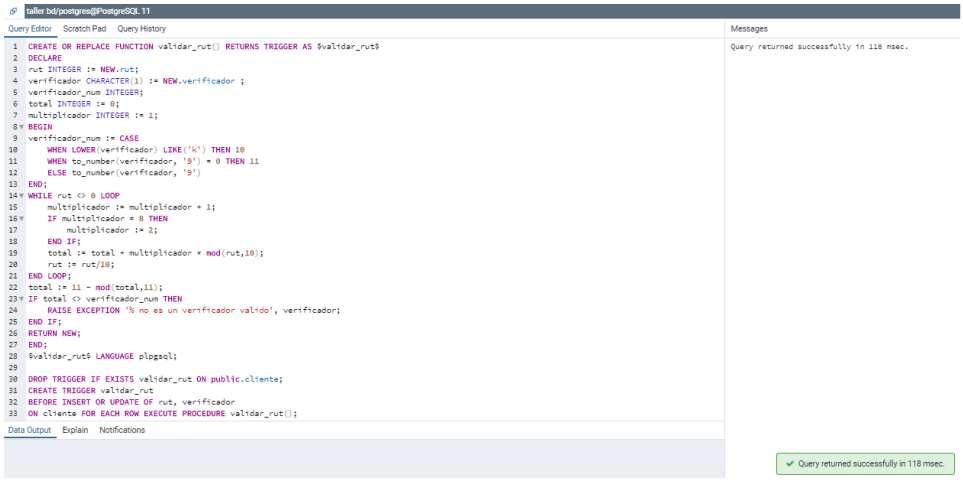
```
1 crear procedimiento comprobarFecha()
2 si nuevo Articulos.auditoria  es nulo entonces
3     lanzar error
4 sino entonces
5     retornar modificacion
```

Finalmente, se crea el disparador de la forma:

```
1 crear disparador
2 antes de insertar en Articulos
3 ejecutar comprobarFecha()
```

2.- Realice un trigger que al modificar o insertar los rut en la tabla correspondiente a No. Cédula, valide que éstos estén correctos.

En la figura 1 se muestra como se crea un procedimiento que se llamara en un disparador que trabaje los datos solicitados.




```
CREATE OR REPLACE FUNCTION validar_rut() RETURNS TRIGGER AS $validar_rut$
DECLARE
rut INTEGER := NEW.rut;
verificador CHARACTER(1) := NEW.verificador ;
verificador_num INTEGER;
total INTEGER := 0;
multiplicador INTEGER := 1;
BEGIN
verificador_num := CASE
    WHEN LOWER(verificador) LIKE('k') THEN 10
    WHEN to_number(verificador, '9') = 0 THEN 11
    ELSE to_number(verificador, '9')
END;
WHILE rut <> 0 LOOP
    multiplicador := multiplicador + 1;
    IF multiplicador = 8 THEN
        multiplicador := 2;
    END IF;
    total := total + multiplicador * mod(rut,10);
    rut := rut/10;
END LOOP;
total := 11 - mod(total,11);
IF total <> verificador_num THEN
    RAISE EXCEPTION '% no es un verificador valido', verificador;
END IF;
RETURN NEW;
END;
$validar_rut$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS validar_rut ON public.cliente;
CREATE TRIGGER validar_rut
BEFORE INSERT OR UPDATE OF rut, verificador
ON cliente FOR EACH ROW EXECUTE PROCEDURE validar_rut();
```

Figura 1: Ejercicio 2 - Trigger

Para comprobar el correcto funcionamiento se busca un valor para modificar a modo de ejemplo, intentando utilizar tanto rut valido como invalido.

Figura 2: Ejercicio 2 - Ejemplo

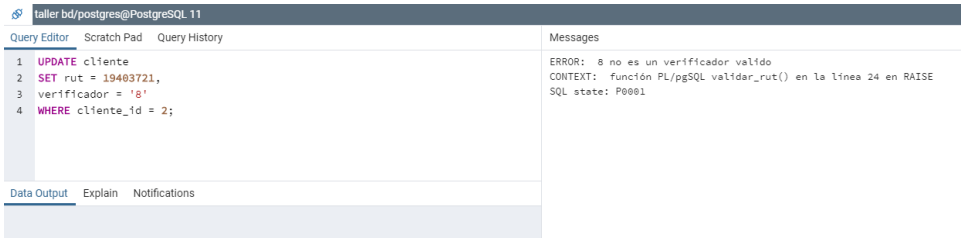


Figura 3: Ejercicio 2 - Error

Figura 4: Ejercicio 2 - Modificación

3.-Realice un trigger que en una tabla denominada “contabilidad”, mantenga el rut del cliente, el apellido paterno y el nombre. Dicha tabla debe ser automáticamente modificada cuando se realice una actualización en el estado actual.

Primero se define la tabla que contendrá los siguientes datos:

- Id de la tarjeta, ya que una persona puede tener mas de una tarjeta.
- Rut.
- Dígito verificador del rut.
- Nombres.
- Apellido paterno.
- Estado actual de la tarjeta.


```
DROP TABLE IF EXISTS contabilidad;
CREATE TABLE contabilidad
(
    tarjeta_id integer NOT NULL,
    rut integer NOT NULL,
    verificador character(1) NOT NULL,
    nombres varchar(200) NOT NULL,
    "primer apellido" varchar(50) NOT NULL,
    "estado actual" estadoactual NOT NULL,
    CONSTRAINT contabilidad_pkey PRIMARY KEY (tarjeta_id),
    CONSTRAINT contabilidad_tarjeta_id_fkey FOREIGN KEY (tarjeta_id)
    REFERENCES tarjeta (tarjeta_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
);
```

Figura 5: Ejercicio 3 - Tabla

Luego, se crea el disparador con el correspondiente procedimiento que se ejecutará cada vez que modifique el estado actual de la tarjeta, con las siguientes consideraciones:

- Contabilidad debe manejar los datos de las tarjetas que se encuentren con algún tipo de deuda, por lo que si se elimina la deuda de la tarjeta, esta debe borrarse de la nueva tabla creada.
- En caso de modificar la deuda a otro valor, solo el campo del estado actual debe actualizarse.

Finalmente se comprueba el correcto funcionamiento del disparador, como se muestra en las figuras 7, 8 y 9.

Figura 6: Ejercicio 3 - Trigger

Figura 7: Ejercicio 3 - Ejemplo

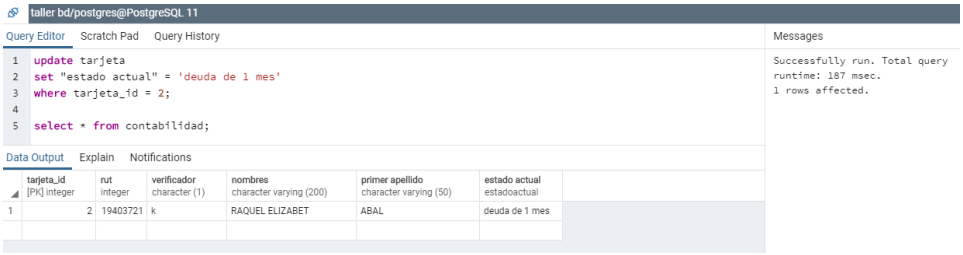


Figura 8: Ejercicio 3 - Sincronización de datos

Figura 9: Ejercicio 3 - Filtrado de datos

3. Conclusiones

Del presente informe se obtuvo en detalle el proceso de organización necesario para conformar la base de datos propuesta para el cliente ficticio, de tal manera que facilita la construcción del software que debe utilizar las variables necesarias para que el negocio funcione de manera exitosa, dejándose expuesta una interpretación clara del modelo definido para entregar tiempos de respuesta breves en futuras consultas.

Además, se obtuvo resultados satisfactorios a la hora de aplicar en su conjunto el conocimiento aplicado en las entregas previas, realización exitosa de modificaciones a la base de datos y la incorporación de nueva materia relacionada a los disparadores de postgresql.