Trabajo Práctico Especial
Bases de Datos I

Fecha de entrega: 12/11/2020

Integrantes:

- D'Ambrosio, David Leandro
- Menjoulou, Mainque Bautista
- Rago, Juan Ignacio

Ayudante designado:

Campos Lozzia , María Cecilia

Índice:

Introducción

Para que se logre un mayor entendimiento, en este informe se explicará y se justificará de manera ordenada por secciones las implementaciones de los diferentes puntos solicitados en el Trabajo Práctico Especial de la materia "**Bases de Datos I"**, usando como referencia el lenguaje de programaciòn PostgreSQL.

Sección A: Ajustes del Esquema

Creación del Esquema

El esquema se creó partiendo del Diseño Lógico Tardío provisto por la cátedra. Usando sentencias tales como CREATE TABLE o ALTER TABLE se logró una representación fiel a lo brindado, manteniendo las claves primarias y extranjeras, los atributos y sus correspondientes tipos.

Tanto las Acciones Referenciales como los controles de matching, no fue necesario agregarlos debido a que no influenciaron en los puntos solicitados del trabajo .

Carga de los datos

Las tablas se llenaron con datos relativamente reales para que las salidas frente a consultas y/o servicios sea más entendible. A continuación se explicará brevemente cómo se fueron llenando cada una de ellas:

- País: se cargaron 20 países, con el nombre en Inglés, y la característica telefónica correspondiente.
- Usuario: se agregaron 10 tuplas correspondientes a usuarios genéricos, con valores de atributos respetando el formato de lo que puede ser un nombre y apellido, e-mail, contraseña, etc. Se asume que todos los usuarios cargados están activos y esto se refleja en el atributo "estado".
- Moneda: respetando la consigna, se insertaron 20 monedas, donde 3 son Fiat , 5 Estables, y las otras 12 criptomonedas. Todas son reales, con sus correspondientes acrónimos y descripciòn. Están activas (reflejado en el atributo *estado*), y en el campo fiat se llena con 'Y' si la moneda lo es, y con 'N' en caso contrario.
- RelMoneda: aquí están las relaciones que hay entre las monedas estables contra las fiat, junto con el histórico. Como esta tabla no se usa durante el trabajo, solamente se agregó una tupla por cada relación.
- Mercado: se insertó un mercado para cada moneda(salvo las fiat) contra las estables, y otro para cada criptomoneda contra el Bitcoin. El nombre del mercado es del tipo *<moneda_origen>_<moneda_destino>* (es decir origen contra destino). El precio del mercado es por defecto 0, y éste se modificará a medida que haya movimientos en la tabla de orden.
- Orden: Se cargaron 100 órdenes, conformadas por 10 usuarios que cargaron una de orden cada tipo (compra, venta) para los mercados BTC_USDT, ETH_DAI, DOT_EURT, BCH_BTC, ADA_BTC. **(1)**
- Billetera: Para cada usuario se cargó una billetera por cada moneda activa en el sistema, con un saldo suficiente para que sus órdenes publicadas de esa moneda tengan fondos en el caso de ser adquiridas.
- Movimiento: En principio se cargaron todos los movimientos de entrada para que un usuario tenga el saldo necesario en sus billeteras (como se explicó en Billetera). Los diferentes tipos de movimientos se basan en la entrada/salida de las monedas en el sistema, estos son "E" y "D", Extracción y Depósito respectivamente. Estos movimientos tienen un bloque y una dirección del blockchain asociados, de acuerdo con la moneda que se haya operado. También están los movimientos generados por la venta/compra, estos son "S", "R", Suma y Resta respectivamente.
- ComposicionOrden: esta tabla no se llena en un principio, porque acá irán aquellas órdenes que se vayan capturando para el servicio de ejecutar una orden de compra y venta. El id de origen indica la orden que se está ejecutando mientras que el id de destino indica la orden que adquiere, mientras que el atributo *cantidad* indica la cantidad que se tomó de la orden destino (si se

adquieren solamente órdenes completas la cantidad será igual a la de la orden adquirida, sino será igual a la cantidad que pueda adquirir el usuario ejecutante).

(1) Planteamos una convención para ver los distintos tipos (Compra, Venta) y estados (Nueva, Ejecutada y Eliminada) de órdenes, veamos un ejemplo con el Mercado: BTC_USDT, donde vendo/compro una **cantidad** BTC a un **valor** de USDT:

Si yo quiero vender una “cantidad” de BTC a un “valor” determinado en USDT, se publica una orden “Nueva” de "Compra", para que otro usuario la compre (yo le vendo mi orden): Es decir, preciso tener en billetera la cantidad de BTC que voy a vender, para que cuando alguien adquiera mi orden, se me descuente de la billetera de BTC y él me de (de su billetera de USDT) el total de mi orden.

Si yo quiero comprar una “cantidad” de BTC a un “valor” determinado en USDT, se publica una orden “Nueva” de "Venta" para que otro usuario me venda sus BTC (yo le doy mis USDT): Es decir, preciso tener en billetera la cantidad de USDT con la que voy a comprar (valor total de mi orden), para que cuando alguien adquiera mi orden, se me descuente de la billetera de USDT y él me de (de su billetera de BTC) la cantidad de mi orden.

Si yo quiero vender una **cantidad determinada** de BTC en USDT, se “ejecuta” una orden del tipo "Compra" que va a estar compuesta por una o más órdenes del tipo "Venta".

Si yo quiero comprar BTC a un **precio determinado** en USDT, se “ejecuta“ una orden del tipo "Venta" que va a estar compuesta por una o más órdenes del tipo "Compra".

Las órdenes adquiridas pasaron de estar en “Nueva” a “Ejecutada” (para ser “Eliminada” se requiere que externamente alguien realice un Delete que afecte a la orden)

Aclaración para B, C, D

No se tuvo en cuenta que las tuplas de ejemplos sean acumulativas, por lo que se pide que preferentemente (antes de seguir probando otros puntos) se borren los cambios que se hayan generado por las tuplas de ejemplo previamente ejecutadas, así el análisis en este informe coincide con lo que se verá en su base de datos.

Sección B: Elaboración de Restricciones y Reglas del Negocio

Regla de Negocio 1

```
ALTER TABLE G02_MOVIMIENTO
    ADD CONSTRAINT chk_G02_1
        CHECK ( NOT EXISTS(
            SELECT 1 FROM g02_movimiento m1
            JOIN g02_movimiento m2 on m2.moneda = m1.moneda
            WHERE (m1.bloque < m2.bloque and m1.fecha > m2.fecha)
            GROUP BY m1.moneda )
    );
```

Captura 1, Regla de negocio 1 en SQL Estándar

Es una restricción de Tabla, por lo que se tuvo que implementar con un Trigger en PostgreSQL, la función del mismo se encarga de llevar la lógica del estándar a procedural. El mismo es activado cuando **se inserta** (after) un nuevo movimiento.

En las tuplas para probar esta RN, la segunda tupla no puede insertarse porque viola la restricción respecto de los datos de la primera inserción. La tercer tupla se inserta correctamente.

Regla de Negocio 2

```
CREATE ASSERTION AS_G02_RN2
    CHECK( NOT EXISTS ( SELECT 1 FROM G02_BILLETERA b WHERE (b.saldo <
      ((SELECT SUM(o.cantidad) FROM G02_MERCADO m JOIN G02_ORDEN o ON m.nombre=o.mercado
            WHERE (o.id_usuario=b.id_usuario AND m.moneda_o=b.moneda AND o.tipo='Compra' AND o.estado='Nueva'))
      +(SELECT SUM(o.cantidad*o.valor) FROM G02_MERCADO m JOIN G02_ORDEN o ON m.nombre=o.mercado
            WHERE (o.id_usuario=b.id_usuario AND m.moneda_d=b.moneda AND o.tipo='Venta' AND o.estado='Nueva'))
      ) ) );
```

Captura 2, Regla de negocio 2 en SQL Estándar

Es una restricción de tipo Assertion, que en PostgreSQL se debe implementar con un Trigger. Se dispara cuando se inserta una orden (ya sea nueva o a ejecutar). Para saber cuanto saldo tengo “reservado” por mis ordenes publicadas de una moneda en especifico, debo sumar todas las órdenes de Compra donde la moneda sea la de origen en el mercado, sumado a la suma total de todas las órdenes de Venta donde la moneda sea la de destino en el mercado, para esto se precisaron variables locales varias consultas de tablas.

En las tuplas de ejemplo en un principio no basta el saldo por lo que no se inserta ninguna, luego de actualizar el saldo al usuario vemos que si volvemos a ejecutar las inserciones, la primera sí cumple ya que ahora hay saldo suficiente pero la segunda sigue sin cumplirse.

Regla del Negocio 3

```
CREATE ASSERTION AS_G02_RN3
CHECK( NOT EXISTS
       ( SELECT 1 FROM G02_MOVIMIENTO mov NATURAL JOIN G02_BILLETERA b
         WHERE ( mov.fecha=(SELECT MAX(mov2.fecha) FROM G02_MOVIMIENTO mov2) and (b.saldo-mov.valor)<
       ((SELECT SUM(o.cantidad) FROM G02_MERCADO m JOIN G02_ORDEN o ON m.nombre=o.mercado
         WHERE (o.id_usuario=b.id_usuario AND m.moneda_o=b.moneda AND o.tipo='Compra' AND o.estado='Nueva'))
       +(SELECT SUM(o.cantidad*o.valor) FROM G02_MERCADO m JOIN G02_ORDEN o ON m.nombre=o.mercado
           WHERE (o.id_usuario=b.id_usuario AND m.moneda_d=b.moneda AND o.tipo='Venta' AND o.estado='Nueva'))
     ) ) );
```

Captura 3, Regla de negocio 3 en SQL Estándar

Es una restricción de tipo Assertion, que en PostgreSQL se debe implementar con un Trigger. Se dispara cuando se inserta un movimiento de extracción Para extraer saldo de una moneda se debe tener suficiente del mismo por lo que se debe calcular cuanto saldo tengo “reservado” al igual que en la RN2.

En las tuplas de ejemplo realiza un ejemplo similar a la regla anterior, pero ahora las inserciones se realizan en movimiento.

Regla del Negocio 4

```
ALTER TABLE G02_MOVIMIENTO
   ADD CONSTRAINT CK_G02_MOVIMIENTO_RN4
       CHECK ( (bloque IS NOT NULL AND direccion IS NOT NULL) OR
                (bloque IS NULL AND direccion IS NULL) );
```

Captura 4, Regla de negocio 4 en SQL Estándar

Es una restricción de tupla, que permite que la implementación en PostgreSQL sea igual que la declaración Estándar.

En las tuplas de ejemplo, la primera no cumple la restricción porque tiene un bloque y no dirección, y la segunda procede ya que ambos son nulos.

Sección C: Servicios

Servicio 1.a

Se implementó una función que dado un mercado por parámetro, nos retorne el precio_mercado del mismo. Para probar el servicio se dejo comentado una consulta a la función misma para un mercado que tiene ordenes.

Servicio 1.b

Se implementó un procedimiento que dado un *id_ejecutante*, un *valor_maximo*, un *tipo_orden* y *mercado_orden* nos ejecutará la orden creada en base a estos parámetros. Sobre la tabla "Orden" en la sección A se explicó cómo se lleva a cabo la ejecución de una orden, para esto se debieron usar varias variables para mantener datos tales como el valor que le queda al usuario ejecutante para gastar, el valor que está adquiriendo de las órdenes y otros auxiliares. A través de un FOR se obtiene la orden más barata de *compra* o la más cara de *venta* (según sea la orden ejecutante) y se verifica si se puede obtener la orden en su totalidad o ya debe terminar. En caso de que se pueda, se inserta en *Composicionorden* la orden ejecutante, la adquirida y el monto de la adquirida. Luego se insertan movimientos de suma y resta para el usuario de la orden de la iteración actual, según corresponda. Así su orden pasa a estar "Ejecutada". En caso de que no se pueda obtener en su totalidad se termina la iteración con un EXIT, y se pasa a insertar los movimientos correspondientes al usuario ejecutante, esto se logra gracias a las variables total_acumulado, valor_maximo, valor_restante.

Cabe destacar que se obtiene una orden para el usuario ejecutante en la que se indican los valores con la fecha actual y esta misma es la usada en *ComposicionOrden*, en caso de que su valor máximo a gastar no alcance para adquirir ninguna orden esta tupla nunca llega a entrar.

Como ejemplo primero se hace un update del saldo de un usuario en un moneda determinada para luego hacer un CALL al procedimiento para que ejecute una orden de compra y tome así las órdenes de venta que la satisfagan. Para ver los resultados se hace una consulta sobre las órdenes del mismo mercado y luego también otra sobre composición orden, donde deberían visualizarse los id órdenes que tomo y sus cantidades.

Servicio 1.c

Se implementó una función que a través de *mercado_x* y *fecha_x* retorna una tabla compuesta por las órdenes creadas en ese día ordenadas por la hora, sin importar su tipo o estado. Las tuplas de ejemplo son de consulta del retorno de la función con diferentes mercados y fechas.

Servicio 2.a

Para mantener actualizado cada precio de mercado se implementó una función que se encarga de actualizarlo mediante auxiliares, que nos permiten guardar el 20%(según las cantidades ordenadas por valor) de las órdenes de venta y compra, el resultado de la suma del promedio de venta y de compra, y una función que nos brinda una tabla con las cantidades acumuladas ordenadas por valor (según el tipo puede ser asc o desc) y así obtener el promedio. Luego ya con ambos promedios se puede hacer un update del precio del mercado actual.

La función es activada por un trigger que la dispara siempre que se inserta una orden o se actualiza el estado de una orden “Nueva”.

Cabe aclarar que esto fue comentado en el script de soluciones ya que lo precisamos para mantener actualizados los mercados entonces lo copiamos en el script de carga antes de las últimas inserciones de la tabla orden.

Servicio 2.b

Se realiza un trigger en la billetera cada vez que se inserta un nuevo movimiento en el sistema. Lo que realiza la función asociada a dicho trigger es actualizar el saldo de la moneda, del usuario que intervino en ese movimiento. Si el movimiento es de suma o depósito (que generan aumento en la billetera), se le suma al saldo actual el valor del movimiento, pero con la comisión del 0.5% aplicada. En cambio si es de resta o extracción, solo se le resta el valor al saldo que tenía, sin aplicar comisión. En el trabajo este trigger se termina activando en el Servicio 1.b cuando se realizan los movimientos de las transacciones entre el usuario ejecutante y el usuario de la orden capturada en cada iteración. No se refleja (por cómo están organizados los scripts) en la extracción o depósito, pero es una funcionalidad que puede soportar estos movimientos.

Sección D: Definición de Vistas

Vista 1

En esta vista solo se realiza un select de todos los atributos de *Billetera* ordenados por *id_usuario* y *moneda*. Cumple para ser automáticamente actualizable, por lo que no tiene problemas al hacerse actualizaciones sobre billetera. Como esta vista no tiene filtro, no existe la posibilidad de migración de tuplas, por esto no se agrega a la vista el With Check Option.

Vista 2

Esta vista tiene más complejidad ya que requiere hacerse la cotización de cada saldo en BTC y USDT, por lo que en la implementación decidimos hacer tres consultas unidas por UNION en las cuales se obtienen los saldos diferentes monedas(y sus cotizaciones), en la primera están todas las billeteras de criptomonedas que están como origen en un mercado contra BTC o USDT (excepto por BTC), en la

segunda están todas las billeteras de monedas estables que están como destino en un mercado con BTC, y por último están todas billeteras de BTC.

No es actualizable por el UNION y por los dos atributos derivados de la proyección. Por lo tanto decidimos que la única operación propagable al aplicarse sobre la vista sea el DELETE, ya que indicaría borrar esa billetera, y si te tienen los datos necesarios.

Vista 3

Esta vista consigue, de la vista 2, la suma del valor en BTC del saldo de cada billetera de un usuario cotizado en USDT (usando la función que obtiene el precio de mercado) y haciendo LIMIT para obtener los diez con más saldo en USDT.

No es actualizable porque uno de los atributos proyectados es derivado, ya que obtiene la suma de todos los saldos en dólares para cada usuario. Al igual que en la Vista 2 decidimos que el DELETE sea la única operación propagable, generando que se borre el usuario, sus billeteras cargadas y las órdenes que pertenecían a dicho usuario pasan al estado de “Eliminada”.

Sección E: Modificación del Esquema

Lo que se hizo para permitir que al ejecutar una orden se pueda adquirir incompletas y así generar otras con el remanente, fue crear una nueva versión (v2) del procedimiento del servicio 1.b con algunas modificaciones respecto a su anterior.

El único cambio fue a la hora de verificar si el *valor_restante* alcanza para adquirir la orden, ahora revisamos que alcance y sobre. En el caso en el que no sobre (es decir sea igual o mayor) se procede a un nuevo bloque. Este bloque se encarga de tomar lo que le alcance de la orden, hacerle los movimientos al usuario de la orden adquirida, marcarla como “Ejecutada” y por último antes de salir con el EXIT se crea una nueva orden con el remanente con los valores respectivos. Fuera del FOR ya se sabe que *valor_restante* es cero, así que también vemos un pequeño cambio en los movimientos del usuario ejecutante.

Conclusión

Este trabajo nos permitió abarcar los principales temas de la materia de Bases de Datos I, desde la creación de un esquema a partir de tablas y las relaciones que poseen entre sí, hasta la implementación de servicios que requieren uso de SQL procedural para automatizar las funcionalidades. Además logró que aprendamos sobre el mundo de las criptomonedas, destacando conceptos principales sobre los mercados, exchange y Blockchain. La gran dificultad del trabajo fue que en la etapa de carga no se tuvieron en cuenta ciertas consideraciones que impactaron en la complejidad de los códigos de la secciones finales.