Trabajo Especial de Bases de Datos I

Grupo 9
Mauro Galvan - 248961
Alejandro Lagar - 248948
Juan Ditella - 248920

Índice

A) Ajuste del esquema

Carga de tablas:

Debe tener por lo menos 10 usuarios.

Para cumplir con este inciso, cargamos 10 usuarios con las ID 001-010 con datos aleatorios y utilizando nombres ficticios a excepción de los 3 primeros.

El Sitio debe manejar 20 monedas y al menos tiene que tener 3 monedas Fiat y 5 estables.

En este punto cargamos exactamente 3 monedas Fiat (Euro, Dólar, Yen), 5 monedas estables (Tether, USD Coin, DAI, Binance USD y Paxos Standard) y las restantes 12 criptomonedas (Bitcoin, Etherum, XRP, Tron, Waves, Tezos, Cosmos, Bitcoin Cash, ChainLink, Binance Coin, Polkadot y Litecoin)

Crear un mercado de cada moneda contra las estables

Se utilizó un procedimiento donde mediante un doble FOR se crean los mercados de cada moneda contra las monedas estables. Se utilizó la función RANDOM para simular los valores de las monedas entre sí. Luego, dentro del segundo WHILE, se procedió a insertar los valores dentro de la tabla Mercado. (también se omite el BTC porque se asocia con estas monedas a continuación)

Crear un mercado de cada criptomoneda contra el BTC

Para lograrlo volvimos a utilizar un procedimiento que utilizando un FOR que selecciona cada moneda que tenga como nombre un valor distinto a 'BTC' (Para evitar un mercado BTC-BTC) y luego inserta en la tabla MERCADO cada una de las combinaciones posibles entre cada moneda y el BTC.

Inicializar las órdenes con al menos 100 filas

En este punto se utilizaron dos procedimientos, el principal pr_g9_cargarOrdenes y un secundario que esta dentro del principal, llamado pr_g9_cargarBilletera.
En el primero se inicia un LOOP que va de 0 a 99, y se agregan 50 órdenes a compra (de la 0 a la 49) y 50 órdenes a venta (de la 50 a la 99).
Luego, se insertan en la tabla Orden, Billetera y Movimiento, en la primera tabla se inserta directamente, mientras que en las restantes se inserta mediante el procedimiento secundario nombrado anteriormente. En dicho procedimiento se agrega en las tablas Billetera los valores id_usuario, moneda y saldo, mientras que en la tabla movimiento se agrega id_usuario, moneda, fecha, tipo, comision, valor, bloque y dirección.

B) ELABORACIÓN DE RESTRICCIONES Y REGLAS DEL NEGOCIO

1) *Controlar que los números de bloque sean consecutivos para los movimientos de Entrada y Salida del Blockchain por moneda y fecha.*

Para la realización de este punto se utilizó una función, que se ejecutará cada vez que se quiera crear un movimiento o actualizar un bloque dentro del movimiento.
En el código se utilizó un IF en donde se ordena todos los movimientos de manera descendente (por número bloque) y se queda con el más grande, luego, no permitirá que se agregue un movimiento en caso de que el número de bloque nuevo sea menor al número de bloque seleccionado, en cuyo caso, se enviará un error a traves de un RAISE EXCEPTION indicando que el movimiento no puede ser agregado.
Mediante un trigger se ejecuta la función descrita cada vez que se quiere agregar o actualizar un valor en tabla movimiento.

2) *Controlar que no se pueda colocar una orden si no hay fondos suficientes*

En este punto se volvió a utilizar una función. En este caso, utilizamos un IF EXIST y, ingresamos con un INNER JOIN a la tabla mercado accediendo a cada valor que tenga un nombre igual al nombre del mercado de la nueva orden que se quiere agregar y ahí seleccionando los valores moneda_o y nombre.
Después, volvemos a hacer un INNER JOIN, pero esta vez a la tabla Billetera, y accederemos a la misma siempre que la ID del usuario que ejecuta la orden sea igual a la ID de la billetera (Es decir, que se accede a billetera del usuario que ejecuta la orden) y además que el nombre de la moneda de origen del mercado (primer INNER JOIN) coincida con la moneda de la billetera a seleccionar.
Una vez encontrada una billetera coincidente, en el caso de que el valor de la orden (el valor se toma por una unidad de la moneda que se quiere comprar o vender), multiplicado por la cantidad de moneda a comprar o vender, sea mayor a la cantidad en la billetera, o bien que el valor de la orden sea menor o igual a 0, entonces saldrá un mensaje de RAISE EXCEPTION y no se permitirá la ejecución de la orden.
Análogamente, se creó un trigger que ejecuta la función descrita cada vez que se quiere agregar o actualizar un valor en tabla orden. (El saldo considerado, es saldo disponible, es decir, el saldo que el usuario tenga puesto en órdenes no está considerado)

3) *No se pueden hacer retiros de una moneda, si esos fondos están en órdenes activas.*

Para este punto se utiliza una función. Luego un IF EXIST, y dentro seleccionamos una billetera de tabla Billetera, mediante un where con varios AND's, verificaremos que la ID de Usuario dentro del movimiento a crear coincida con la ID de usuario de la Billetera de la cual queremos retirar (Es decir, que el usuario que quiere retirar sea el dueño de la billetera a analizar) , nos fijamos que la moneda que se quiere retirar sea la misma que la moneda de la billetera que estamos analizando. Por último, corroboramos que el valor a retirar sumado a la comisión que nos cobran sea mayor al saldo en la billetera. En el caso que todas estas condiciones se cumplan, entonces el total que se debe descontar de la billetera (retiro + comisión) es mayor a lo que tenemos en la billetera actualmente, por lo que no podremos retirar nuestra moneda. Por lo tanto, sale un mensaje de RAISE EXCEPTION y no se permite el ingreso del movimiento en la tabla.
Adicionalmente, se programó un TRIGGER que se ejecutará siempre que se quiera agregar algo a la tabla Movimiento. (El saldo considerado, es saldo disponible, es decir, el saldo que el usuario tenga puesto en órdenes no está considerado)

4) *La opcionalidad del Número de Bloque en Movimiento, debe coincidir con la opcionalidad de Dirección, es decir que ambos son nulos o ambos no lo son.*

Mediante una función y un IF EXIST, seleccionamos una billetera de tabla Billetera. Seguido a esto, tenemos un WHERE con una condición de OR donde verificamos que ambos datos (Bloque y Dirección) de nuestra nueva Billetera sean NULL o NOT NULL, es decir, que en caso de que sean diferentes entre sí (Uno NULL y otro NOT NULL) salga un mensaje de RAISE EXCEPTION y no se permita ingresar el bloque dentro de la tabla Billetera.

Adicionalmente agregamos un TRIGGER que ejecutará siempre que se quiera agregar algo a la tabla Billetera.

C) SERVICIOS

1) *Utilizando el método que considere más adecuado, implemente un servicio que:*
   A) *Brinde el precio actual de cotización en un mercado determinado.*

Se utilizan dos funciones, una se encarga de calcular cual es el 20% de una moneda en un mercado, tanto para la compra como para la venta y de llamar a la otra función.
Por otro lado, la otra función se encarga de calcular el promedio del valor del 20% menos costoso de las órdenes de compra y venta, mediante un promedio entre estas dos obtenemos el nuevo valor de mercado el cual se actualiza con un UPDATE.

B) Ejecute una orden de mercado (orden del tipo Market) para compra y venta.

Se utiliza una función y dos vistas para la realización del ejercicio..

La primera función comienza con un if, que se encarga de ver si la orden es de tipo compra o venta.
luego de esto, mediante unos for recorro todas las tuplas de una vista que cumplan con un par de condiciones, como que el mercado de las órdenes sea el requerido, que estén sin cumplir, y que el valor de la orden sea igual o menor al que ofrezco para la compra, o igual o mayor para la venta.
las órdenes son tomadas de menor a mayor valor por unidad, descontando el valor de las tomadas del total que quería gastar o ganar, y marcado las órdenes tomadas como cumplidas. por último carga en las billeteras lo correspondiente a las órdenes cumplidas, (te suma la moneda que ganaste y te resta la que diste a cambio),finalmente te devuelve un remanente del porcentaje que no se haya podido cumplir de tu orden (por ejemplo, si quisiste comprar 100 y había 80 a la venta, te devuelve el equivalente a los 20 faltantes, en la moneda que estabas dando a cambio). el funcionamiento para la compra o venta es similar, cambiando algunos detalles.

Esta función es llamada por otra, que se encarga de pasarle los datos de la orden a la anteriormente mencionada, y de setear la orden que pasa como cumplida.

Por último, un trigger es activado, en caso de que una orden nueva ingrese, o una antigua sea actualizada. .

C) Dado un mercado X y una fecha, retorne un listado con todas las órdenes ordenadas en forma cronológica junto con su tipo y estado (si está cumplida, si es nueva, etc.).

Se utiliza una función y se pasan por parámetros los atributos nombreMercado y fecha_param.

En el procedimiento, se retorna una query que contiene los valores ID, Tipo y Estado ordenado por fecha_creacion y traído de la tabla Orden para algún valor donde el atributo o_mercado sea igual al nombre_mercado y la fecha de creación sea igual a la fecha pasada por parámetro, ordenado según la fecha en la que se creó dicha orden.

D) DEFINICIÓN DE VISTAS

1) *Realice una vista que dé el saldo de cada moneda para cada uno de los usuarios.*

   Seleccionamos de Billetera los valores ID_Usuario, Moneda y Saldo y mostramos todos los valores cargados ordenados por ID_Usuario

2) *Realice una vista que dé el listado de la cantidad de dinero que tiene cada usuario en cada moneda junto con la cotización al momento de ejecutar la vista en BTC y USDT.*

   Utilizamos una vista que no es actualizable, por ende, tuvimos que generar un trigger instant of para su inserción, actualización y borrado.

   En la vista seleccionamos los valores ID_Usuario, Moneda, Saldo, Moneda_D y Precio_Mercado de la tabla Billetera, entrando a la tabla Mercado.
   seleccionando los valores Moneda_D, Precio_Mercado y Moneda_O cuando se cumpla que la moneda_d tenga como nombre BTC o USDT y que además la moneda_o de mercado sea igual al nombre de moneda de billetera.

   En la primera función, la de inserción y actualización, rfn_9_insertUpdate_SaldoUsuarios(), se verifica mediante dos if que la función sea de inserción o de actualización, luego, se inserta en la billetera siempre que no exista una billetera para el usuario nuevo.
   Si se decide actualización, entrará en el segundo if, donde se actualizarán los valores de la billetera acorde a los nuevos valores del usuario que desea actualizar.

   La segunda función se dedicara a eliminar una billetera en caso de requerirse

3) *Realice una vista (usando la vista anterior) que dé los 10 usuarios que más dinero tienen en TOTAL en toda su billetera.*

   En esta vista seleccionamos de la tabla SaldoUsuario_BTC_USDT todos los valores donde la moneda destino sea BTC o la moneda de origen sea BTC, luego se agrupan por ID_Usuario y se ordenan por saldoprecio_mercado seleccionando los 10 mayores (se utilizó BTC para que la suma de dinero total sea toda en una sola moneda).

E) MODIFICACIÓN DEL ESQUEMA

Para solucionar este problema decidimos reutilizar el procedimiento fn_g9_ejecutarOrden, esta función se encarga de ejecutar la orden solicitada tomando órdenes del tipo inverso (Si la solicitada es compra, venta o a la inversa), a su vez, actualizaba las billeteras del cliente del cual se ejecutó su orden y la de los clientes que completaron su orden para cumplir la del anteriormente mencionado. Luego de esto, la orden se seteaba como completa, la modificación que realizamos fue en vez de marcar la orden como 'Completa' actualizar el valor de la misma dejando como valor el restante del de la orden inicial menos el porcentaje que se obtuvo, es decir, si queria 100 bitcoins y logró comprar 80, se genera una nueva orden por el 20 restante, como el enunciado indica que no modifique servicios anteriores, la función no se ejecuta en nuestro programa, ya que al ser llamada, activa el trigger que se encarga de ejecutar una nueva orden insertada, o una vieja orden actualizada, por ende llamarla modifica servicios.

CONCLUSIÓN

En conclusión, el trabajo nos sirvió para poder entender cómo funcionan las bases de datos de forma práctica. Comenzando por la creación de tabla y sus restricciones, el uso de SQL Procedural, la creación de restricciones, reglas de negocio, la creación y utilización de funciones y sus triggers de ejecución, así como también la utilización a fondo de las vistas y optimización de consultas.
Pudimos también comprender de mejor manera todos los conceptos nombrados, así como su relación entre ellos.
Como punto extra, pudimos también aprender mejor del lenguaje PostgreSQL y la utilización del DataGrip.