Trabajo Especial de Bases de Datos I

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

Indice

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, Dolar, 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 utilizo un procedimiento donde mediante un doble FOR se crean los mercados de cada moneda contra las monedas estables. Se utilizo la funcion RANDOM para simular los valores de las monedas entre si. Luego, dentro del segundo WHILE, se procedio a insertar los valores dentro de la tabla Mercado.

Crear un mercado de cada criptomoneda contra el BTC

Para lograrlo volvimos a utilizar un procedimiento que utilizando un FOR que selecciona cad a 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 ordenes a compra (de la 0 a la 49) y 50 ordenes 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 utilizo una funcion, que se ejecutara cada vez que se quiera crear o reemplazar un nuevo bloque dentro de movimiento.
En el codigo se utilizo un IF que ordena todos los movimientos de manera descendente y se queda con el mas grande, luego, no permitira que se agregue un movimiento en caso de que el numero de bloque nuevo sea menor numero de bloque seleccionado, en cuyo caso, se enviara un error a traves de un RAISE EXCEPTION indicando que el movimiento no puede ser agregado.
Analogamente, se creo un trigger que ejecuta la funcion descripta 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 seleccionamos una orden desde la tabla orden. Luego, 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 al a billetera del usuario que ejecuta la orden) y ademas 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 sea mayor a la cantidad en la billetera, o bien que el valor de la orden sea menor o igual a 0, entonces saldra un mensaje de RAISE EXCEPTION y no se permitira la ejecucion de la orden.
Analogamente, se creo un trigger que ejecuta la funcion descripta 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 ordenes no esta considerado)

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

Para este punto se utilizo otra vez una funcion. Luego, utilizamos otra vez un IF EXIST, y dentro seleccionamos una billetera de tabla Billetera. Despues, 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) luego, nos fijaremos que la moneda que se quiere retirar sea la misma que la moneda de la billetera que estamos analizando. Por ultimo, corroboramos que el valor a retirar sumado a la comision 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 + comision) 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 ordenes no esta 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.*

Para este punto se utilizo otra vez una funcion. Luego, utilizamos otra vez un IF EXIST, y dentro seleccionamos una billetera de tabla Billetera. Seguido a esto, tenemos un WHERE con una condicion de OR donde verificamos que ambos datos (Bloque y Direccion) de nuestra nueva Billetera sean NULL o NOT NULL, es decir, que en caso de que sean diferentes entre si (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 ejecutara 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 funcion.
Por otro lado, la otra funcion se encarga de calcular el promedio del valor del 20% menos costoso de las ordenes 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 funcion y dos vistas para la realizacion del ejercicio.
En la funcion se pasan por parametro las funciones ingreso, mercado1 y tipo_orden, adicionalmente se declaran el integer suma y el record Rec.
En el procedimiento se utiliza un IF para saber si la orden es o no es de tipo 'Venta'. En caso de serla, se inicia un FOR que finalizara cuando la suma de las compras sea mayor al Ingreso (que se pasa por parametro) y se seleccionan todos los valores de la tabla Orden_Venta donde el nombre del mercado sea igual que el mercado pasado por parametro, y donde el estado sea 'Pendiente'.
Luego se hace update a todas las ordenes y se sete el estado a 'Cumplida' en el caso de que la id del record sea la misma que la id de la venta que estamos seleccionando, por ultimo, se suma la cantidad de la orden que se selecciono.

En caso de que el tipo orden no sea venta, se inicia un FOR que finalizara cuando la suma de las compras sea mayor al Ingreso (que se pasa por parametro) y se seleccionan todos los valores de la tabla Orden_Compra donde el nombre del mercado sea igual que el mercado pasado por parametro, y donde el estado sea 'Pendiente'.
Luego se hace update a todas las ordenes y se sete el estado a 'Cumplida' en el caso de que la id del record sea la misma que la id de la venta que estamos seleccionando, por ultimo, se suma la cantidad de la orden que se selecciono.

Es llamada por un TRIGGER para la automatizacion.

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 funcion y se pasan por parametros los atributos nombreMercado y fecha_param. Adicionalmente, se declara un valor numerico.

En el procedimiento, se retorna una query que contiene los valores ID, Tipo y Estado ordenado por fecha_creacion y traido de la tabla Orden para algun valor donde el atributo o_mercado sea igual al nombre_mercado y la fecha de creacion sea igual a la fecha pasada por parametro.

D) DEFINICION DE VISTAS

No utilizamos wait check option porque en el punto 1 no tenemos ninguna condicion del where, por lo tanto, no es necesario utilizarlo

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 y 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 ademas la moneda_o de mercado sea igual al nombre de mercado de billetera.

En la primera función, la de inserción y actualizacion, 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, entrara en el segundo if, donde se actualizaran los valores de la billetera acorde a los nuevos valores del usuario que desea actualizar.

La segunda funcion 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.

E) MODIFICACIÓN DEL ESQUEMA

Para solucionar este problema decidimos reutilizar el procedimiento fn_g9_ejecutarOrden, esta funcion se encargaba de ejecutar la orden solicitada tomando ordenes del tipo inverso (Si la solicitada es compra, venta o a la inversa), a su vez, actualizaba las billeteras del cliente del cual se ejecuto 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 valor de la orden inicial menos el porcentaje que se obtuvo.

CONCLUSION

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, 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.