TRABAJO PRÁCTICO ESPECIAL

*Base de Datos I - 2020*



Grupo *04*

**Integrantes:**

*[249843 - Coria, Santiago]*

*[248589 - Fumiatti, Matias]*

*[248530 - Santos, Matias]*

# 

# Índice

[**Índice**](#_zi4joddt2ygz) **2**

[**A. Ajuste del Esquema**](#_ddse01w3rys1) **3**

[**B. Elaboración de Restricciones y Reglas del Negocio**](#_3bvfyzrzva9r) **4**

[B.1](#_qvvfaa7v90el) 4

[B.2](#_owp2qx4sfqj) 4

[B.3](#_iwv7xesfcv37) 4

[B.4](#_a00rzixrfxqd) 5

[**C. Servicios**](#_patef1s79cqc) **5**

[C.1.a](#_nbxsp9y9xbgw) 5

[C.1.b](#_wfukz4kw87ta) 5

[C.1.c](#_pwt7g3ko5od8) 6

[C.2](#_v2epum57bane) 6

[**D. Vistas**](#_tfecbmdaj11t) **6**

[D.1](#_byaxwxlz8u3t) 6

[D.2](#_im53ur9s3n23) 6

[D.3](#_9ipih3tqrlqx) 7

[**E. Modificación del esquema**](#_oimrgbsvagld) **7**

[**Conclusión**](#_b85is8nwijal) **8**

# A. Ajuste del Esquema

No se realizaron modificaciones en el esquema provisto por la cátedra más allá de renombrar las tablas, PKs, FKs, etc. con el respectivo nombre de nuestro grupo con lo cual proseguimos a realizar la creación de los esquemas con sus RIR’s correspondientes por medio de Vertebelo.

Para insertar las monedas y los mercados realizamos los insert de manera manual, comenzando por moneda, para cada una de las tablas, con sus respectivas características solicitadas en el enunciado.

Se decidió no utilizar la tabla Rel\_moneda ya que no nos aportará nada a la implementación de las soluciones que se plantearon.

Para la carga de la Base de Datos insertamos manualmente un único país que es Argentina para que la creación de los 100 Usuarios, a su vez también usamos una función para generar fechas aleatorias para variar la fecha de alta de cada usuario.

Al mismo tiempo cuando se inserta un usuario se activa un trigger que va a recorrer la tabla de monedas y por cada una de ellas le creará a dicho usuario una billetera con saldo 0.

Para hacer el llenado de la tabla Orden se hizo un procedimiento el cual por cada mercado va a llamar a otro procedimiento que recorre todos los usuarios e inserta una cantidad de órdenes pasada por parámetro, de tipo compra o venta. Previo a la inserción en la tabla Orden, se le actualiza el valor de la billetera de la que se quiere extraer saldo. Cabe destacar, que utilizamos el parámetro “tipo” para decidir si la orden era de compra o venta de la moneda\_o.

Si bien no se inicializa la tabla de Movimiento, tomamos la consideración de que para los movimientos que se realizan gracias a la ejecución de alguna orden, le asignamos el tipo “t”(transacción) al movimiento entre dinero de usuarios, y para los movimientos de entrada o salida de dinero del blockchain le asignamos “e” o “s” respectivamente.

Finalmente para la tabla Composicion\_orden tomamos como id\_d la orden que se llama a ejecutar(que se verá en el apartado C.1.b) y como id\_o, la/las órdenes que se tengan que ejecutar para complir con el llamado de la función.

# 

# B. Elaboración de Restricciones y Reglas del Negocio

## B.1

Para SQL estándar se planteó utilizar un check de tabla, el cual realizará una consulta de not exists en la que revisaremos que no exista una tupla en la cual tanto el bloque o la fecha insertada sea menor a los valores de esa tupla.

Ya que PostgreSQL no nos permite realizar subconsultas dentro de un check, en el primer apartado utilizamos SQL procedural. Se implementa un trigger en el cual por cada insert en Movimiento revisará que la tupla a insertar cumpla con las restricciones impuestas anteriormente. Para revisar el correcto funcionamiento de dicho trigger, se plantea realizar varios inserts en movimiento. Primero se inserta un movimiento de testeo, el cual utilizaremos para comprobar las comparaciones y los raise exceptions se ejecuten correctamente. Luego se intentan insertar 2 movimientos distintos, el primero de ellos nos dará error porque la fecha ingresada es menor al máximo de la tabla, y el segundo, porque el número de bloque es igual al máximo de la tabla y debe de ser mayor estricto.

## B.2

Para la resolución de esta RIR, decidimos utilizar un assertion en SQL estándar ya que debemos consultar valores en más de una tabla, el cual comprobará que el usuario dado tenga saldo suficiente para poder realizar esta orden, ya sea de venta o de compra en el mercado dado.

Como PostgreSQL, no soporta utilizar assertions, se plantea una solución mediante la implementación de 2 triggers, uno para la compra y otro para la venta, los cuales se llamarán cuando se inserte o se actualice tanto el campo valor o cantidad de una tupla en Orden. Las funciones que se ejecutan si bien son similares porque ambas extraen el saldo, tienen la diferencia de que cuando se quiere insertar una orden de compra se debe calcular que se tenga el saldo suficiente mediante la multiplicación de new.valor \* new.cantidad, y cuando se inserta una de venta, solo se requiere chequear por new.cantidad ya que el valor no importa para este caso.

Para comprobar el funcionamiento se borrarán las órdenes en un mercado y el saldo de la billetera se modificará a 100 para el usuario 1, y luego intentaremos hacer un insert de compra que supera su saldo, al intentar hacer esto nos dirá que el usuario no tiene saldo suficiente.

Una vez comprobado el insert para compra, se comprueba que para insertar una venta tenga saldo de la moneda a vender, para esto le modificamos el saldo de la moneda a 1 y intentamos insertar una orden de venta de 2, la misma no se podrá ejecutar ya que no tiene el suficiente saldo.

## B.3

Para la resolución de esta RIR, decidimos utilizar un assertion en SQL estándar ya que debemos consultar valores en más de una tabla, el cual comprobará que la moneda que se quiere extraer, no figure en ningún mercado de la tabla ordenes(para esto deberemos pasar antes por Mercado para extraer todos los mercados que matcheen con la moneda que se quiere extraer), donde estén en estado activo, y su usuario coincida con el usuario quiere extraer saldo.

Como PostgreSQL, no soporta utilizar assertions, se plantea una solución que realiza lo mismo pero utilizando funciones de trigger, en la cual, en vez de consultar por la no-existencia de una tupla, preguntaremos por la existencia de la misma, y en caso de ser así, lanzaremos un raise exception.

Para chequear el correcto funcionamiento de este trigger, comenzamos eliminando todas las órdenes de un usuario específico para eliminar errores inesperados, luego se inserta una orden de tipo venta, y finalmente al querer insertar un movimiento de tipo ”salida”, nos arrojará un error, ya que tenemos una orden activa en el mercado, para esta moneda.

## B.4

Para la última restricción utilizamos SQL estándar declarativo, donde tenemos una restricción de tipo de tupla en la cual verificamos con un check que tanto dirección como bloque sean ambos atributos nulos o no.

Para el chequeo del correcto funcionamiento del check, se intentan insertar 2 movimientos, los cuales no cumplen con la restricción del check, ya que alguno de los 2 valores son null y el otro no.

# C. Servicios

## C.1.a

Para llevar a cabo este servicio, creamos una vista que posee una columna donde ya tiene calculado el valor por cantidad y así poder realizar cálculos de manera más sencilla.

Para el desarrollo de este servicio dividimos el problema en varias funciones para lograr una mejor comprensión de la resolución, ya que debimos realizar varios cálculos que, en un solo procedimiento, resultaban muy tediosos de analizar. Comenzamos con la función principal, que recibe como parámetro el mercado que se requiere consultar para obtener el precio actual de la cotización. Luego de esto, llamamos a *calculaTotalCrypto(varchar(20), varchar(10))*, la cual le pasamos el mercado y si es compra o venta, esta nos retorna la suma de la cantidad de crypto que se tiene actualmente en las órdenes de ese mercado, lo cual nos deja con 2 valores, uno para compra y otro para venta. Luego multiplicamos este valor por 0.2 para obtener el 20% de la cantidad, y finalmente recorremos las órdenes en el orden correcto(asc para venta y dec para compra) hasta llegar a la cantidad requerida. Finalmente, calculamos el promedio simple entre estos 2 valores, y lo retornamos.

## C.1.b

Para realizar esta solicitud, implementamos un procedimiento ejecutarOrden(bigint) el cual, a su vez, llama a 2 procedimientos distintos, dependiendo si la orden es de compra o venta, ya que hay algunas diferencias de funcionamiento. En caso de que no exista una orden con ese id, o el id de la misma sea distinto a “compra” o “venta”, se lanzarán las respectivas excepciones. Luego, en caso de que todas las condiciones comprobadas no arrojen ningún error, obtendremos como resultado, la modificación, si es que es posible de la orden dada a estado finalizado, con las órdenes del tipo contrario que sean necesarias también finalizadas. En caso de que no se haya podido completar la orden entera, se le devolverá el saldo restante al usuario que realizó la orden. El llamado a esta función también, de ser necesario, insertará tuplas en las tablas Composicion\_orden y Movimiento, mediante las cuales podremos ver el historial de cómo se fueron ejecutando tanto las órdenes como los movimientos.

## C.1.c

Para este servicio desarrollamos una función que recibe por parámetro un mercado y una fecha. Con dichos parámetros verificamos que el mercado sea el mismo que recibimos por parámetro y que la fecha de las órdenes se encuentre entre la fecha que recibimos por parámetro y la fecha actual. La respuesta a esta función es devuelta en forma de una tabla que tiene los datos solicitados en el enunciado.

## C.2

Para la resolución de este servicio al tener que mantenerse actualizados por sí mismos, se implementan 2 funciones de trigger respectivamente, las cuales llaman a la función 1.a y 1.b por separado con sus respectivos parámetros. Los triggers serán activados al momento de insertar una orden en la tabla Orden, o en el caso del 1.a cuando se realiza una modificación en los atributos de “estado” o “fecha\_ejec” de alguna de las tuplas de dicha tabla.

# D. Vistas

## D.1

Esta vista es automáticamente actualizable tanto para SQL estándar, como para PostgreSQL, ya que estamos retirando únicamente valores de una sola tabla, y, a su vez, todos los datos necesarios para establecer la PK de las tuplas. Al utilizar la opción de chequeo “with check option” no generaría ningún cambio, ya que en la vista no estamos filtrando por ningún valor, por lo cual siempre seremos capaces de realizar cualquier tipo de modificación de la tabla, ya sea con inserts, updates o eliminación de tuplas

Cabe destacar que para la resolución de este ejercicio se planteó que sólo se tuviera en cuenta el saldo que se tiene en la billetera, ya que el saldo que se tiene en las órdenes lo consideramos como saldo congelado.

## D.2

Esta vista no resulta actualizable ya que para mostrar todas las columnas se deben realizar cálculos internos para poder devolver los saldos en BTC y USDT, por lo cual tuvimos que recurrir a un trigger instead of, para lograr que la vista sea actualizable. Además de esto, cuando levantamos esta vista, estamos realizando 2 joins con otras vistas auxiliares, lo cual también genera que la misma no sea actualizable. Para esto, establecimos una regla del negocio por la cual, si se quiere insertar saldo en una billetera, tanto la columna de BTC como la de USDT deben ser nulas, a la hora de hacer un insert.

## D.3

Al momento de plantear esta vista utilizando la anterior, se tomo como que los usuarios más ricos se calcularán a través de la columna saldo en BTC, una vez aclarado esto lo que se llevó a cabo fue por agrupar por usuario, así podremos realizar la proyección del “id\_usuario” y la suma de todas las monedas en BTC.

Como la vista no resulta actualizable se desarrollan trigger instead of para los servicios de insert,update y delete.

Para el insert se plantea que si existe el usuario le haga un update de la billetera BTC con el saldo insertado, en caso de no existir el usuario daría error, para la sentencia de update se decidió poner en 0 todo el saldo de las billeteras del usuario y que solo tenga saldo en BTC por la cantidad a modificar. Por último en el delete se utilizó el atributo de “estado” en la tabla usuario en el cual al activarse el trigger marcará al usuario como inactivo.

# E. Modificación del esquema

Para poder explicar esta resolución nos basaremos en un ejemplo. Supongamos que el usuario 1 quiere comprar 1 BTC a un precio de 15000 USDT. Para esto comenzamos ordenando las órdenes de venta de este mercado de menor precio a mayor. Luego empezamos a ejecutar cada orden de forma procedural, hasta llegar a, por ejemplo, una orden de venta de un usuario 2, que quiere vender 3 BTC, a un precio de 13000 USDT c/u. Ahora supongamos que la orden de compra lleva 0.75 BTC acumulados, que han sido comprados de órdenes anteriores a la del usuario 2. Como podemos observar, todavía faltan 0.25 BTC para poder concretar la orden del usuario 1, y la del usuario 2 excede esa cantidad, por lo cual se requiere dividirla en 2 más chicas. Para esto, lo que se hace es pasar la orden de venta original a estado finalizada, y marcarla con fecha\_ejec como current\_timestamp, y luego de esto, tomar el resto de la orden para generar una nueva(con cantidad = 2.75), que, a su vez, activará el trigger nuevamente, generando una reacción en cadena donde se ejecutarán varias órdenes hasta que se estabilice el mercado.

# 

# Conclusión

Gracias a la realización del trabajo se puedo aplicar todos los conocimientos obtenidos a lo largo de la cursada y comprender mejor el funcionamiento de un blockchain.

Además al ser algo que constantemente se está actualizando y maneja grandes volúmenes de datos, se planteó desde un principio el hacer la resolución de los servicios de la forma más eficiente posible.

A lo largo del desarrollo del trabajo tuvimos que tomar muchas decisiones sobre implementación que si bien es difícil el poder consultar solo a una persona, entendemos que el dia de mañana si se nos da la posibilidad de trabajar de DBA, vamos a tener un PO al cual poder consultarle cómo quiere que se realicen determinados servicios ante determinados escenarios.

Por último, pero no menos importante, destacamos que a medida que la cursada avanzaba con los temas dados, debimos adaptar nuestro trabajo para poder llegar a soluciones mejor logradas, tanto por eficiencia, como por facilidad de lectura del desarrollador.