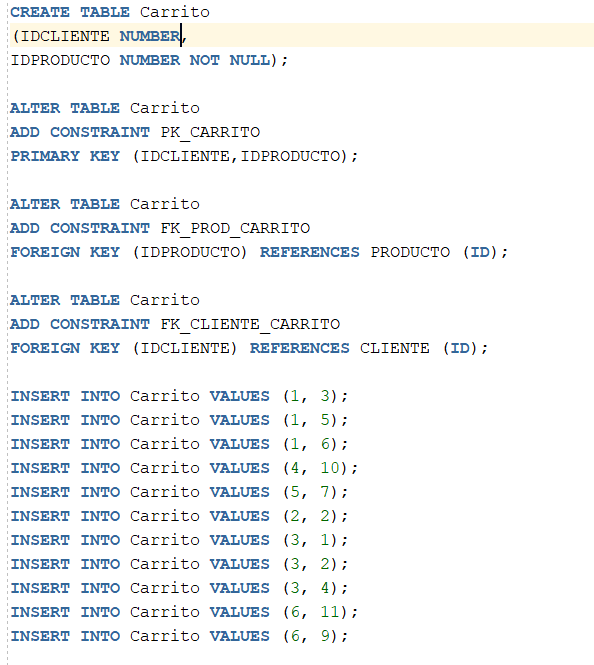
**Documento de análisis y diseño iteración 3**

**Cambios modelo conceptual**

**Cambios clase carrito**

Desde la primera iteración, por las descripciones dadas en el documento de contexto del caso, habíamos visionado el uso de un carrito para las compras, por lo que al realizar los primeros modelos conceptual y relacional del proyecto, se incluyo la entidad (clase) y la tabla de carrito, sin embargo a estos no se les dio uso debido a que las indicaciones de la segunda iteración proponían una forma distinta a la inicialmente planteada para el pago de productos, por lo que el concepto de carrito era innecesario. Para esta tercera iteración se propuso finalmente el uso de un carrito para el proceso de compra, sin embargo las indicaciones a seguir que detallaban su uso nos obligaron a replantearnos la implementación.

Aquí podemos observar la implementación original en la base de datos:



Con la descripción original pensamos que la mejor forma de implementar un carrito era asociando el id de un producto que cogía el cliente con su id. Sin embargo nos dimos cuenta que esta implementación causaba varias complicaciones, debido a que no se podía saber que cantidad del producto cogía el cliente, y para los requerimiento funcionales de esta iteración, no se podían cumplir un numero de funciones: abandonar un carrito, coger un carrito vacio, devolver los productos en un carrito, etc.

Considerando lo anterior, decidimos que era mejor implementarlo de esta manera:

-Colocarle un id al carrito, para que esta tenga independencia del cliente y por lo tanto pueda ser abandonado, re asignado,etc.

-Dejar la tabla solo con el id del carrito y el id de su actual ocupante, si no se tiene ocupante se colocará el id del cliente en 0, que es la forma de representar que no pertenece a ningún cliente.

-Crear otra tabla llamada contiene carrito, la función de esta tablaes llevar el registro de los productos que se tienen en el carrito, las columnas de la tabla son el id del carrito, el id del producto y la cantidad que lleva de este producto, de esta forma un carrito puede tener varios productos al mismo tiempo y a si mismo puede haber una cantidad mayor a uno de un producto especifico.

La implementación nueva quedo así:

Imagen que contiene Texto

Descripción generada automáticamente

**Cambios pedido.**

Para esta iteración se propuso cambiar la forma en la que se realizan los pedidos, para la iteración pasada, simplemente cuando un gerente de sucursal quería realizar un pedio o cuando se realizaba automáticamente debido a que se alcanzaba el nivel de reorden de un producto, solo tocaba esperar a la llegada del pedido para concretar el proceso. En esta nueva fase del proyecto se propuso un nuevo paso, consolidar los pedidos. Básicamente con este nuevo paso las ordenes de pedido que se hacían en la iteración anterior ahora tienen que agruparse por proveedor para una sucursal, de forma que el gerente tiene que utilizar la aplicación para consolidar los pedidos, es decir hacerlos realmente ya que las ordenes no se solicitarán al proveedor hasta que haya una orden de pedido consolidado.

Esta era la forma en la que se hacia previamente a esta iteración:

Imagen que contiene Texto

Descripción generada automáticamente

Considerando el análisis hecho previamente decidimos que para poder construir el nuevo proceso de pedio propuesto era necesario mantener la tabla de pedidos pero agregando una nueva tabla para los pedidos consolidados.

Debido a que las ordenes de pedido se generan antes de hacer un pedido consolidado, realmente no tienen varias de las características que se les había asginado originalmente, como fecha esperada, fecha de llegada, estado y calificación, estas ahora pertenecerán a el pedido consolidado, debido a que este es el verdadero pedido que se le realiza al proveedor.

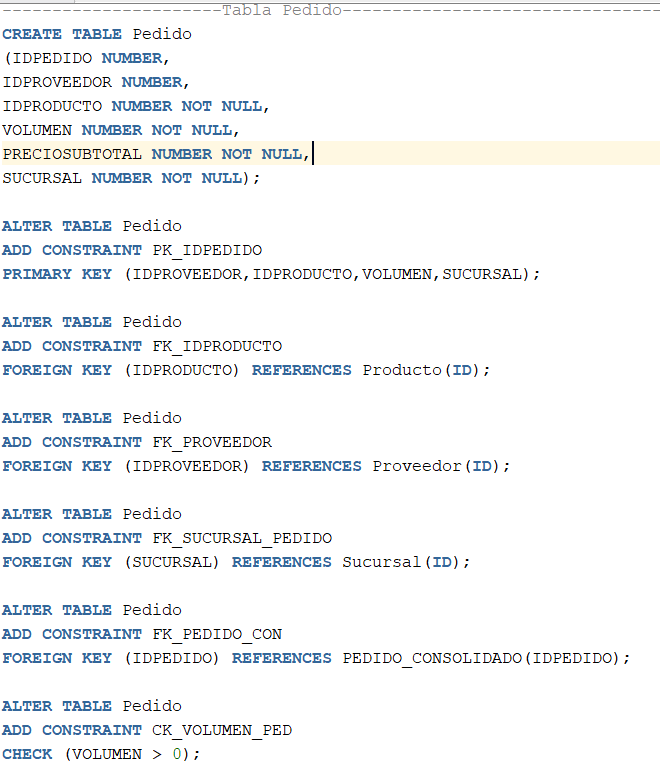
Realizamos puntualmente los siguientes cambios en la tabla de pedido:

-Va a tener un id de pedido que va a corresponder al pedido consolidado al que pertenece, como las ordenes de pedido se crean antes de que se consolide el pedido, al crearse la orden se le va a asignar el id de pedido 0, que no corresponde a ningún pedido consolidado, el id se va a cambiar una vez esta orden haga parta de un pedido consolidado. Es importante que se asocie una orden de pedido con un pedido consolidado ya que una vez llegue el pedido, se tiene que saber que se incluía en el pedido consolidado.

-Adicionalmente a lo anterior se mantienen las columnas de idproveedor, idproducto, volumen, precio sub total y sucursal. Todos estos datos son necesarios para la que se pueda realizar los procesos de consolidar pedio y llegada de pedido consolidado.

Las nuevas tablas quedan asi

**Pedido**



**Pedido consolidado**

Tabla

Descripción generada automáticamente con confianza baja

**Listado de tablas con sus metadatos**

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente con confianza media

**Documentación lógica aplicación**

**RF15**

La implementación de este requerimiento empieza en la interfaz, donde al ejecutarse le pedirá al usuario (cliente) su correo, el input ingresado por el usuario se pasara a la función solicitarCarrito de la clase de lógica de superAndes

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

Una vez en el método se llamara a el método de la persistencia darClientePorCorreo(), pasando por parámetro el correo obtenido en la interfaz, este método devolverá un objeto de la clase cliente, que tendrá toda la información del cliente que esta solicitando el carrito. Se llamara al método getId() del objeto para conseguir el id del cliente. Se llamara al método de la persistencia darCarritosDisponibles() que devolverá todos los carritos existentes que no tengan asignado un cliente, el resultado del método se guardara en una lista de objetos de la clase carrito,si no existe ningún carrito libre se devolverá una lista vacia.

Texto

Descripción generada automáticamente

Posteriormente con un if se evaluara si la lista esta vacia (lo que quiere decir que no hay carritos libres), en caso de cumplirse la condición se llamara al método de la persistencia adicionarCarrito(), que creara un nuevo carrito en la base de datos sin asignarle un cliente, el método devolverá un objeto de la clase carrito. Se llamara al método getId del objeto para obtener el id del carrito que acabamos de crear , posteriormente se llamara al método de la persistencia asginarCarrito() pasando por parámetro el id del carrito creado y el id del cliente, el metododo hará cambios en la base de datos para que el carrito creado se encuentre asignado al cliente que lo esta solicitando.

Texto, Carta

Descripción generada automáticamente

En caso de que la condición del if no se cumpla, quiere decir que si hay carritos disponibles, por lo tanto se llamara al método get de la lista de carritos, pasando por parámetro el numero 0 que hará que se obtenga el primer carrito de la lista, llamando al método get id del carrito obtenemos su identificador y llamamos al método limpiarCarrito pasándolo por parámetro, este método va a realizar la función de devolver los productos a los estantes si el carrito disponible aun tiene estos. Finalmente se llamara a la función asignar carrito con el id del cliente y el id del carrito.

Texto, Carta

Descripción generada automáticamente

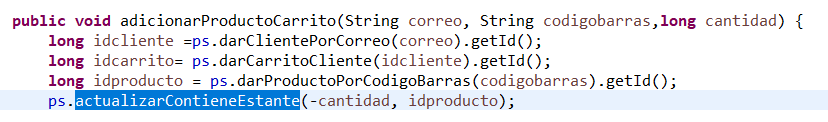
**RF16**

La implementación empieza en la interfaz con el método adicionarProductoCarrito(), el método pide en la interfaz el correo, el código de barras del producto y la cantidad que se va a agregar al carrito, si estos datos no son nulos se va a llamar a el método del mismo nombre en la clase de lógica de la aplicación, pasando los 3 datos pedidos por parámetro

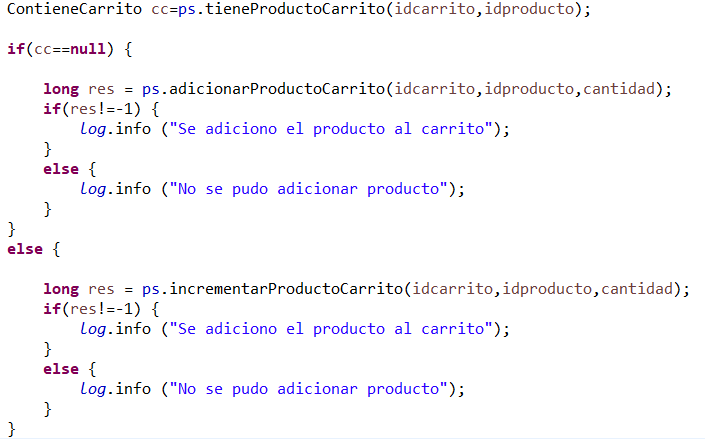
Texto

Descripción generada automáticamente

En el método de la clase de lógica se va a obtener el id del cliente llamando al método de la persistencia darClientePorCorreo() y llamando el método getId sobre este método. Con el id del cliente se va a llamar al método de la persistencia darCarritoCliente() pasandolo por parámetro, este método nos va a devolver el objeto Carrito que correpondera al carrito que esta utilizando el cliente, se llama al método get id sobre el resultado para obtener el id del carrito. Con el código de barras obtenido por parámetro se llama al método de la persistencia darProductoPorCodigoBarras() y se obtiene el id de este. A continuación se llama al método de la persistencia actualizarContieneEstante() pasando por parámetro la cantidad obetenida por parámetro de manera negtiva y el id del producto, esto hará que el método quite la cantidad cogida del producto del estante donde se encuentra.



Posteriormente se llamara al método tieneProductoCarrito() de la persistencia pasando por parámetro el id del carrito y el id del producto, este método nos devolverá la entrada en la tabla de datos donde esta el carrito utilizado y el producto que se va agregar, si el método devuelvo el objeto nulo quiere decir que en el carrito no se ha agregado el producto anteriormente, si devuelve un objeto ContieneCarrito quiere decir que el producto ya se encuentra en el carrito entonces solo se debe aumentar la cantidad. Lo anterior se evalua con un if y se ejecutan una de las 2 opciones dependiendo del resultado.



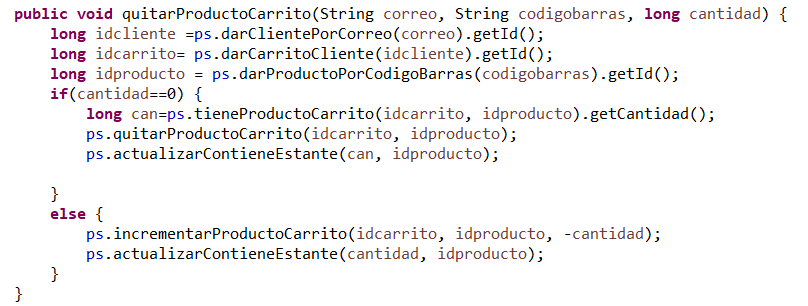
**RF17**

En la interfaz se empieza pidiendo por input al usuario su correo, el código de barras del producto que desea quitar del carrito y la cantidad del producto que desea quitar, se aclara que si se desea quitar todas las unidades del producto que se encuentran en el carro se coloca 0 cuando se pregunta la cantidad. Posteriormente se llama a la función de la clase de lógica quitarProductoCarrito con los datos pedidos.

Imagen que contiene Gráfico

Descripción generada automáticamente

En el método en la lógica se llama al método de la persistencia darClientePorCorreo y a getId de este resultado para obtener el id del cliente, se obtiene el id del carrito del cliente por medio del metodo darCarritoCliente(idcliente).getId() y el id del producto llamando al método darProductoPorCodigoBarras(codigobarras).getId(). A continuación, se analiza si se paso un 0 como cantidad para eliminar todos los productos, esto por medio de un if. Si la cantidad si corresponde a 0 se llamara al método de la persistencia tieneProductoCarrito que nos devolverá la cantidad del producto que se encuentra en el carrito, se llamara al método quitarProductoCarrito, con el id del carrito y el id del producto, esto alterara la base de datos para que elimine el producto del carrito, siguiente a esto se llamara al método actualizarContieneEstante con la cantidad obtenida del producto y el id del producto, esto hará que en la base de datos se refleje que al estante del producto devuelto se devolvieron todas las unidades que se encontraban en el carrito. Si la cantidad es diferente de 0 quiere decir que se va a eliminar una cantidad especifica, se va a entrar al else del if y se llamara al método incrementarProductoCarrito, pasando el id del carrito el id del producto y la cantidad de forma negativa, lo que hará que en la base de datos se refleje la disminución en la cantidad de unidades que se encuentra en el carrito del producto. Finalmente se llamara a la función actualizarContieneEstante.



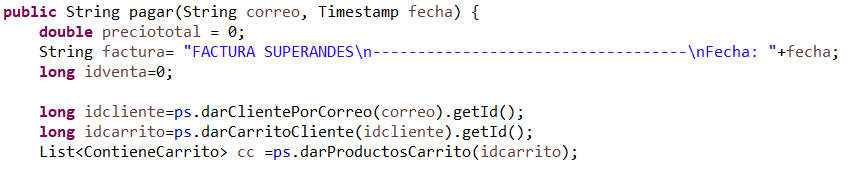
**RF18**

La implementación empieza en la interfaz con el método pagar(), donde se pide al usuario por input el correo del cliente que realiza la compra y la fecha actual. Los datos se pasan por parámetro en la llamada al método del mismo nombre en la clase de lógica.

Texto

Descripción generada automáticamente

En el método de lógica se inicia la variable preciototal en 0, en esta se va a llevar la cuenta final a pagar por el cliente, también se inicializa la variable factura para dar el inicio al mensaje de factura que se devolverá, se inicia también la variable idventa en 0. Con el correo pasado por parámetro se obtiene el id del cliente al llamar a darClientePorCorreo(correo).getId(), y se obtiene el id del carrito con ps.darCarritoCliente(idcliente).getId(), con el id del carrito se van a obtener todos los productos que se encuentran en este llamando al método de la persistencia darProductosCarrito. Que nos va a devolver una lista de objetos ContieneCarrito, donde cada objeto tiene el id del producto y la cantidad que se encuentra en el carrito.



A continuacion se va a iterar sobre la lista con los productos del carrito, se obtiene el Objeto ContieneCarrito y se sacan los valores de la cantidad y el id del producto, se llama al método

ps.darProductoPorId(idproducto).getCantidad() que nos va a dar la cantidad actual del producto, vamos a llamar a ps.darProductoPorId(idproducto) para obtener el objeto producto si la cantidad que se va a comprar es mayor a la cantidad que se tiene del producto no se va a poder ejecutar la venta del producto por falta de inventario. Si por el contrario la cantidad es menor a la cantidad actual se va a actualizar el inventario del producto llamando a ps.actualizarProducto(-cantidad, idproducto), posteriormente se van a conseguir los datos de bodega del producto para ver si es necesario ejecutar una orden de pedido por el nivel de reorden.

Texto

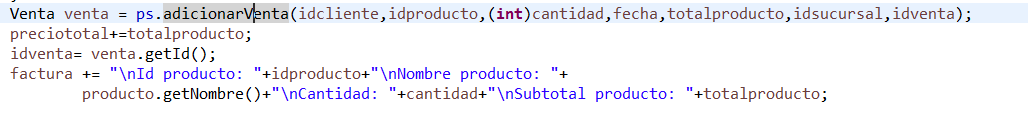
Descripción generada automáticamente

Si el nivel de reorden es mayor a la cantidad actual de l producto se llama a el método adicionarPedido() con los datos correspondientes para hacer la orden de pedido. Posteriormente se analiza si existen promociones existentes para el producto y se aplican los respectivos descuentos en caso de que se tengan.

Texto

Descripción generada automáticamente

Se obtiene el precio total de la compra del producto y se llama al método ps.adicionarVenta() para agregar la venta del producto a la tabla de ventas. Se agrega el subtotal del producto al contador que se inicio en el proceso y se agrega los datos de la venta del producto a la factura, posteriormente continua el ciclo hasta que se procesan todos los productos en el carrito.



Finalmente se limpia el carrito de los productos y se abandona.

Texto

Descripción generada automáticamente

**RF19**

En la interfaz se va a pedir el correo del cliente que va a abandonar el carrito, con su correo se llama al método ps.darClientePorCorreo(correo), lo que nos va a devolver un objeto Cliente del cual sacaremos el id de este, con el id se obtendrá el carrito que tiene asignado, con el id del carrito se llamara al método ps.asignarCarrito(carrito.getId(), 0), se pasa el 0 por parámetro para que el método haga que en la base de datos se refleje que el carrito no tiene ningún cliente asignado.

Texto

Descripción generada automáticamente

**RF20**

En la interfaz por input se pide el id del carrito del cual se van a devolver los productos, con este llamamos a la función darProductosCarrito() de la persistencia que nos va a devolver todos los productos que posee el carrito con su cantidad, se itera sobre cada producto y se obtiene el id y la cantidad que había en el carrito, con estos datos se llama a la función actualizarContieneEstante() de la persistencia que va a actualizar en la base de datos la cantidad que se va a encontrar en los estantes de un producto con estos devueltos. Finalmente después de terminar el ciclo se llama a la función limpiarCarrito de la persistencia que va a borrar de la base de datos la asociación del carrito con los producto que tenia.

**Texto

Descripción generada automáticamente**

**RF21**

Por la interfaz se pedirá por input el correo de la persona que intenta realizar el proceso, su clave y la fecha esperada para el pedido que se va a consolidar. Con el correo y la clave se darán 3 escenarios, en el primero la contraseña dada no corresponde con la contreaseña que se tiene en la base de datos del usuario, por lo que se acabara el proceso en ese momento. Otro caso es que el correo y contraseña si correspondan al usuario, pero el rol de este no sea gerente de sucursal por lo que no estará autorizado y se acabara ahí el proceso. En el ultimo caso el rol del usuario si corresponde a gerente de sucursal, se procederá a crear una lista vacia para meter los ids de los pedidos de la sucursal. Se obtendrá el id de la sucursal del usuario y se llamara al método darPedidosPorSucursal() que devolverá una lista con las ordenes de pedido de esa sucursal. Se crear una lista vacia donde posteriormente se agregaran los id de los proveedores de los pedidos. Por medio de un ciclo, agregamos todos los proveedores a la lista obteniéndolos de la lista de ordenes de pedido. Posteriormente iteramos sobre la lista de ids de proveedores, para cada proveedor se van a obtener los pedidos específicos de ese proveedor para esa sucursal por medio del método darPedidosPorSucursalYProveedor(idsucursal, proveedores.get(i)). Vamos a iterar sobre estos pedidos obteniendo el precio de cada uno y agregándolo a un contando del precio final del pedido consolidado. Se procede a crear el pedido consolidado con sus datos con el método ps.consolidarPedido(), que además devolverá el id del pedido consolidado, con el cual procederemos a asginar el id para las ordenes de pedido por medio del método asignarPedidoConsolidado()

