“Reporte técnico iteración 2”

Cristhian Peña Jaimes 201714313,

Francisco González Rey 201713245

Profesor: Germán Bravo

Universidad de los Andes, Bogotá, Colombia

{fj.gonzalez, cc.pena[}@uniandes.edu.co](mailto:%7d@uniandes.edu.co)

Fecha de presentación: octubre 3 de 2018

Tabla de contenido

[1 Introducción 1](#_Toc529088879)

[2 Análisis. 2](#_Toc529088880)

[3 Diseño de la aplicación. 3](#_Toc529088881)

[3.1 Impacto en el modelo conceptual. 3](#_Toc529088882)

[3.2 Listado de tablas. 3](#_Toc529088883)

[3.3 Validación de tablas en BCNF. 5](#_Toc529088884)

[3.4 Requerimientos funcionales. 5](#_Toc529088885)

[4 Construcción de la aplicación. 7](#_Toc529088886)

[4.1 Ajuste en la aplicación a partir de los nuevos requerimientos. 7](#_Toc529088887)

[4.2 Verificación de comportamiento transaccional. 7](#_Toc529088888)

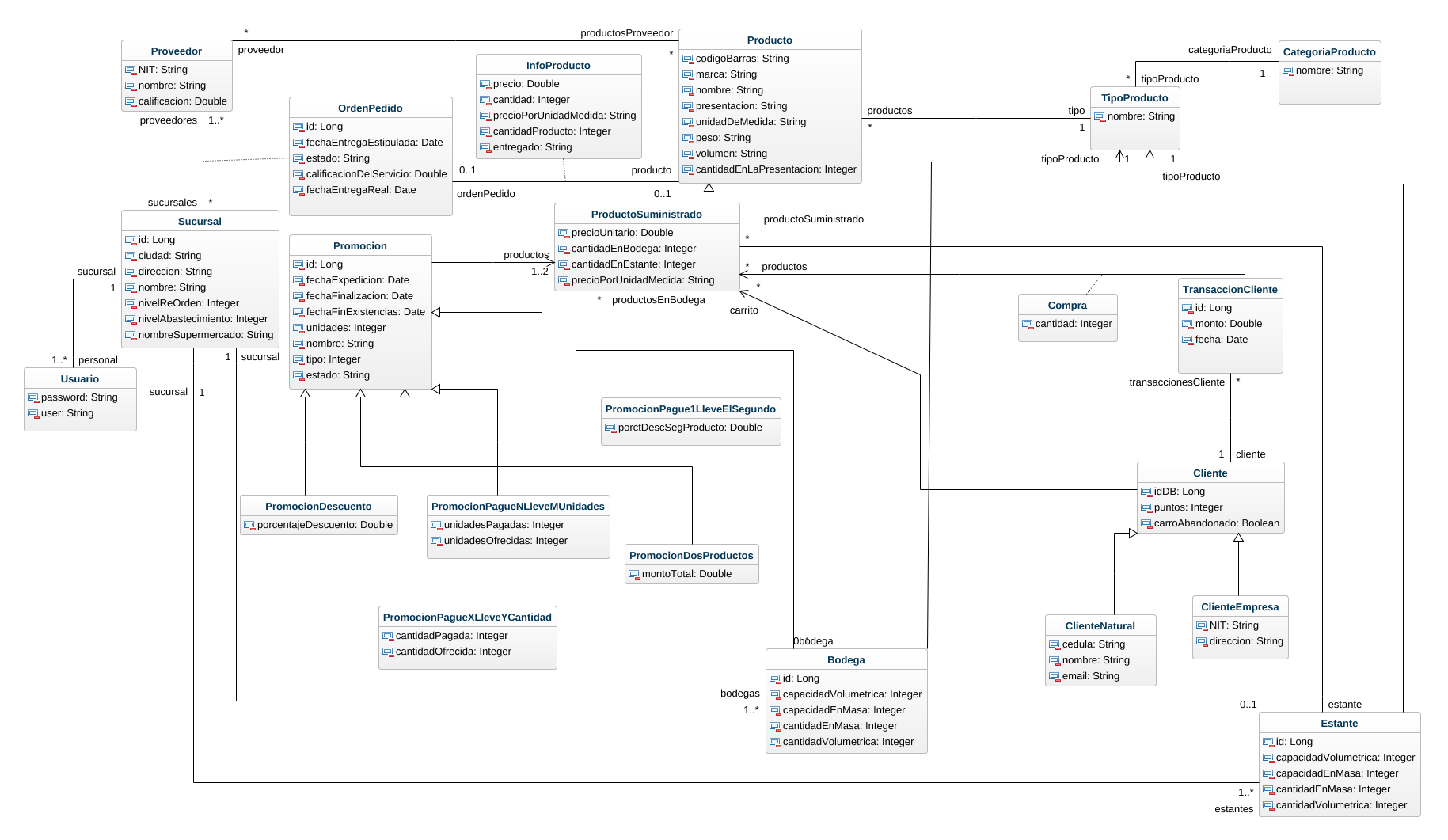
# Introducción

Este es un documento que lleva el reporte del trabajo desarrollado en la iteración 2. A partir del siguiente, se espera poder comprender, realizar y completar los requerimientos solicitados que permitan integrar los conocimientos aprendidos en clase a aplicaciones de la vida real.

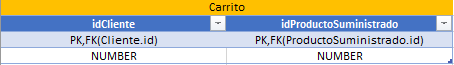
La aplicación que se implementará en este caso es SuperAndes, con dicha aplicación y sus respectivas reglas de negocio se realizarán los siguientes ítems. En primer lugar, nos centraremos en corregir los errores y fallas obtenidos a partir de la anterior implementación. Estas correcciones parten desde el análisis de la forma normal de nuestras tablas. En segundo lugar, nos enfocaremos en la resolución de los nuevos requerimientos funcionales y en la verificación ACID de dichos requerimientos. Para fines de calidad de implementación se desarrollarán pruebas a cada uno de los requerimientos, mediante una réplica de la aplicación original llamada "Demo" la cual está programada para ejecutar cada uno de los métodos de la aplicación y validar su buen funcionamiento. Además, se adjuntarán las sentencias SQL y sus respectivos resultados.

Para finalizar, este documento será el respaldo del proyecto desempeñado en Java junto con su documentación respectiva y las sentencias SQL desarrolladas en el software sqlDeveloper.

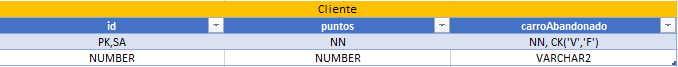
# Análisis.



*Imagen 1. Modelo conceptual actualizado.*



*Imagen 2. tabla carrito.*



*Imagen 3. tabla CLIENTE.*

A partir de la necesidad yaciente de implementar los requerimientos ligados al carro de compras, hemos decido crear la tabla carrito. Está tabla es razón de una asociación entre el cliente y sus productos en el momento que desea comprar. En otras palabras, estos productos persisten para el cliente mediante este desee realizar la compra; por ende, las decisiones que toma sobre que añadir y sacar del carro repercuten en la tabla nueva con finde guardar registro. De la misma manera, para abandonar el carro se simplifica dicho proceso debido a que el cliente tendrá un nuevo atributo que indicará si abandonó o no su carro. Basados en esta idea un cliente sólo puede tener un auto simultáneamente.

Por otra parte, para implementar las ordenes de pedido consolidadas se agregará un nuevo tipo de estado a dichas ordenes que se llamará *EN\_PROCESO.* Esto nos permite ser más flexibles a la hora de manejarla ya que al emitir una orden dada una compra del cliente está queda *EN\_PROCESO*. Después, el personal de una sucursal determinada “consolida” las ordenes (cambiar su estado a CONSOLIDADA) y se las “envía al proveedor”.

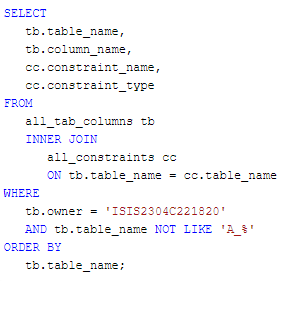
# Diseño de la aplicación.

## Impacto en el modelo conceptual.

El impacto generado en el modelo conceptual, el modelo relacional y la lógica del problema es significativo debido a que este requerimiento implica la conexión persistente y concurrente de los productos del usuario. Además, el hecho de persistir esos datos implica modificaciones lógicas. A la hora de crear una transacción para el cliente previamente se debe finalizar el proceso de agregar/eliminar productos del carro. Por otra parte, la introducción de ordenes pedido consolidadas genera un cambio en la implementación lógica previa. Mientras en la iteración 1 realizábamos ordenes cada vez que el reorden se incumplía, ahora se debe esperar y a partir de una “consolidación” realizarla. Este requerimiento se ajusta más a la realidad y permite reducir problemas de transaccionalidad entre las sucursales y los proveedores.

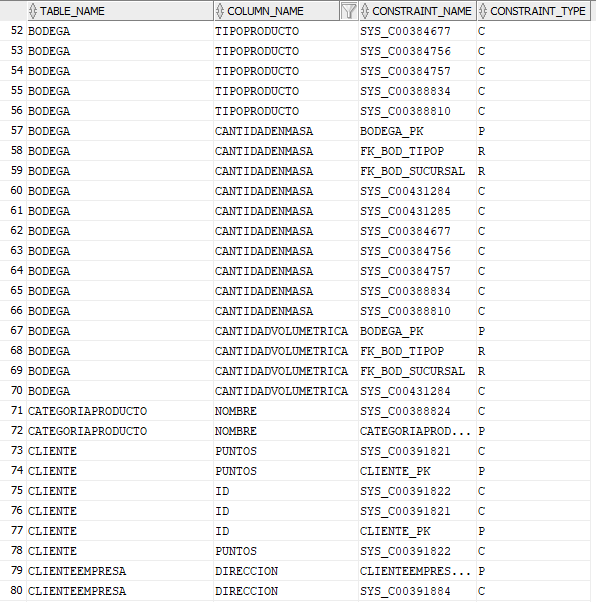
## Listado de tablas.

Esta fue la sentencia SQL creada para obtener el listado de todas las tablas.

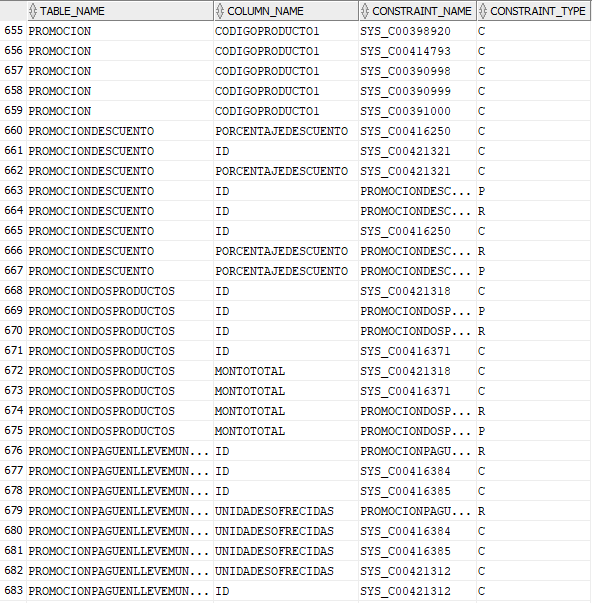


*Imagen 3. SeNTENCIA SQL PARA OBTENER TODAS LAS TABLAS.*

Estos son algunos de los resultados obtenidos:



*Imagen 4. Ejemplo n.1 del resultado de la sentencia.*



*Imagen 5. Ejemplo n.1 del resultado de la sentencia.*

## Validación de tablas en BCNF.

Para validar la forma normal de las tablas aplicaremos los conceptos básicos instruidos en clase.

1*NF:* Todas las tablas construidas se encuentran en primera forma normal. Esto se debe a que no existen atributos multivalor en ninguna de ellas.

2*NF:* Se puede afirmar que todas las tablas construidas se encuentran en segunda forma normal ya que se encuentran en primera forma normal y no tienen dependencias parciales.

3*NF:* Para estar en tercera forma normal se debe estar previamente en segunda forma normal y cumplir con el concepto de no tener dependencias transitivas. Este concepto no existe ya que ningún atributo no primo tiene una relación directa con otro no primo en las tablas.

BC*NF:* Al tener tablas con atributos primos simples y estar en 3NF podemos asegurar que estamos en BC. Incluso, no existen dependencias parciales, es decir que para todo (X 🡪 Y), X es una super llave.

## Requerimientos funcionales.

A continuación, se describirán los mecanismos empleados para asegurar las propiedades ACID. Al final se hará una breve explicación de como se asegura que dichas propiedades se cumplen también en los requerimientos de la iteración anterior.

* **RF12-RF13-RF14. Solicitar un carrito de compras, adicionar un producto al carrito de compras y devolver un producto del carrito de compras.**

A partir de la creación de la tabla carrito, mencionada con anterioridad, logramos satisfacer varios de los principios de ACID. El hecho de que en dicha tabla se registre los productos que se agregan y/o sacan del carro permita satisfacer la consistencia y atomicidad buscadas, es decir, persistir la información en las tablas permite garantizar que la información a la cual se le ha hecho *commit* no se pierda. Esto implica, directamente, satisfacer dicha durabilidad buscada ya que la tabla guardará todo lo añadido. Incluso, se satisface la atomicidad porque para todos los clientes que deseen realizar una compra van a tener registro de su carro de compras y la realización de sus transacciones. A su vez, el agregar y eliminar productos lo realizamos basados en estas propiedades. Al suplir la necesidad de que la información consultada sea verídica cuando un cliente va a comprar ve cuantas unidades del producto determinado hay actualmente. Esta vista es consecuencia de que a la hora de agregar y eliminar se actualiza la cantidad en estante en la tabla del PRODUCTOSUMINISTRADO.

* **RF15.Pagar la compra.**

Vale la pena decir que cada vez que se agrega un producto se actualiza su cantidad en estante. Esto se debe a que queremos que la aplicación le permita al usuario saber cuanta disponibilidad hay de un producto x, es decir, si no se actualizase los usuarios verían información falsa que complicaría el momento de la compra. Teniendo claro esto a la hora de realizar la compra solo se procede a registrársela a ese cliente y facturar.

* **RF16. Abandonar un carro.**

Bajo la premisa de que un cliente maneja un y sólo un carro simultáneamente se crea el atributo “carroAbandonado” que busca tener registro de dicha decisión de abandonar. Debido a que abandonar no es sinónimo de devolver al instante los productos, se busca tenerlo en un estado temporal para que cuando el operario de la sucursal quiera devolver los productos le quede más sencillo.

Las propiedades ACID se garantizan con este atributo. Todos pueden abandonar un carro al tiempo y al tener cada uno un carro se maneja aquel aislamiento buscado. Además, garantizamos que cuando se haga commit de dicho abandono este persista en la tabla del cliente. Por ende, abandonar es un simple *udpate* de condiciones en la tabla cliente y no implica(aún) meternos con una debida actualización en estantes.

* **RF17. Recolectar productos abandonados.**

Las propiedades ACID se garantizan para este requerimiento debido a que este proceso es realizado por el personal de la sucursal sobre los carros de los clientes con estado abandonado. Además, buscamos hacer commit cada vez que se hace update de un producto y no commit por cada recolección ya que así logramos evitar problemas con durabilidad ni consistencia y satisfacer el aislamiento. En pocas palabras, una vez la sucursal decida recoger los productos se procede a devolver a todos, sin excepción.

* **RF18. Consolidar pedidos a los proveedores.**

En búsqueda de las propiedades ACID seguimos con la línea de crear un orden pedido cada que se realiza una compra. La diferencia fundamental es que estas ordenes no se emiten directamente al proveedor, sino que se dejan en estado temporal de en proceso y ya posteriormente se genera dicha consolidación[[1]](#footnote-1) de las órdenes. A su vez, el aislamiento que se busca es de sucursales, ósea que cada personal de sucursal pueda realizar dicho proceso con las ordenes de su sucursal. Tenemos que destacar que dicha consolidación toma las ordenes en proceso asociadas a un proveedor y las comprime en una sola. Esta “compresión” implica que las ordenes en proceso se terminan eliminado y persiste una sola que tiene por estado CONSOLIDADA.

* **RF19. Registrar llegada de pedido consolidado.**

La llegada de este pedido es el cambio en el estado de la orden. Los pedidos se registran manualmente por los operarios de la sucursal. Las ordenes pedido actualizadas tienen por estado ENTREGADO. Posteriormente, se procede a registrar lo suministrado por los proveedores en bodegas y estantes. El proveedor juega un papel muy importante en este requerimiento ya que es el quién decide cuando “enviar” los productos. Esta acción la hace mediante el cambio de estado de la orden a EN\_CAMINO lo cual le permite a la sucursal saber que dicha orden está por llegar.

# Construcción de la aplicación.

## Ajuste en la aplicación a partir de los nuevos requerimientos.

Teniendo en cuenta lo mencionado con anterioridad, en esta iteración se decidió crear la clase Carrito que hace referencia a la asociación entre un cliente y producto suministrado. Por esta razón, las transacciones respectivas a comprar tuvieron un cambio importante, ya que ahora las compras de un cliente se realización solamente por el carro. En contraste, la iteración 1 contaba con un mecanismo similar pero no persistente de compra ya que el cliente no tenía conocimiento de lo que llevaba.

Por otra parte, la consolidación de ordenes pedido implico la inclusión de un nuevo estado en la clase OrdenPedido y la modificación de generación de dichas órdenes. Como fue mencionado, las ordenes siguen persistiendo, pero su estado es el que nos permite como aplicación manejar todas las operaciones transaccionales.

## Verificación de comportamiento transaccional.

Para realizar la verificación del comportamiento transaccional de las aplicaciones se implementaron pruebas JUnit. Estas pruebas contemplan la posibilidad de que una transacción sea exitosa o no. Esta dualidad se da gracias a la variedad de datos suministrados, ya que se selecciono con detalle aquella información que haría fallar o no las operaciones funcionales. Es así como nos aseguramos de que los requerimientos satisfagan las necesidades estipuladas y sigan con la lógica planteada a lo largo de este documento.

El documento Excel que lleva registro de estar pruebas se encuentra en la carpeta *docs* bajo el nombre de “*informacionTest*”.

1. Esta consolidación es sinónimo de la creación de una nueva orden que resume lo de las ordenes añadidas de ese proveedor y dicha sucursal. [↑](#footnote-ref-1)