

“Proyecto del curso: SuperAndes – Iteración 2”

Juan F. Torres Piza, Nicolás Cobos Carranza

El propósito de este documento es recopilar el análisis y resultados obtenidos a través del desarrollo (segunda parte) del modelaje e implementación de la aplicación de un supermercado, SuperAndes. Este documento muestra paso a paso el desarrollo de la segunda iteración del proyecto desde los cambios en los modelos diseñados hasta la implementación de la base de datos teniendo en cuenta las reglas ACID, de la aplicación y de sus pruebas respectivas.

Universidad de los Andes, Bogotá, Colombia

{jf.torresp, [n.cobos](mailto:n.cobos@uniandes.edu.co)}@uniandes.edu.co

Fecha de presentación: Noviembre 4 de 2018

Tabla de contenido

1	Introducción	1
2	Cambios en los modelos	2
2.1	Modelo Conceptual	2
2.2	Modelo Relacional	2
3	Resultados	4
3.1	Resultados logrados	4
3.2	Resultados no logrados	8
4	Balance plan de pruebas	8
5	Análisis reglas de negocios	8
6	Conclusiones	9
7	Bibliografía	10

1 Introducción

Uniandes ha decidido implementar SUPERANDES, una aplicación que apoye a los supermercados en su operación diaria. Cada supermercado que opera utilizando SUPERANDES tiene su propia aplicación. (tomado del enunciado del proyecto).

El propósito de este documento es el análisis de resultados del proceso de desarrollo de la iteración 2 para la construcción de la aplicación SuperAndes para manejo de supermercados. En este documento se realizará un reporte de todos los resultados obtenidos desde el replanteamiento del modelo conceptual y relacional con los nuevos requerimientos, hasta el balance de pruebas y análisis de las reglas de negocio. El documento se divide en 5 partes: la

Almacenamiento. Esta clase tiene un tipo, ya sea estante o bodega, y esto determinará su identificador, su capacidad de peso y volumen y su cantidad de productos y dependiendo del tipo, se tendrán en cuenta las existencias y el nivel de abastecimiento de bodega. Esto nos facilitó el manejo de datos y la organización de los mismos a la hora de implementar los requerimientos.

Finalmente, se agregó el atributo de estado a la clase Transacción (pagado, pendiente) para facilitar el requerimiento de pago de compra de un carrito y además se modificó la clase Promoción para agregar un tipo con lo que se manejaría el proceso de promoción dependiendo de su tipo (ej. 2x1, 50% de descuento).

Estos cambios fueron modificados en el modelo de *GenMyModel* y también en el modelo relacional y en *Java*.

2.2 Modelo Relacional

Estas fueron las tablas resultantes y modificadas para la iteración 2:

TABLA CARRITO			
ID	Estado	Clave	Sucursal
PK,SA	NN, CK('LIBRE', 'EN USO', 'ABANDONADO', 'PAGADO')	NN, CK (mayor o igual a cero)	FK SUCURSAL.ID

Figura 2. Tabla CARRITO, modelo relacional.

TABLA CONTIENE		
idProducto	Cantidad	idCarrito
FK PRODUCTO.ID	NN, CK (MAYOR A CERO)	FK FACTURA.NUMERO

Figura 3. Tabla CONTIENE, modelo relacional.

TABLA ALMACENAMIENTO								
ID	CapacidadVolumen	CapacidadPeso	Producto	Sucursal	Existencias	CapacidadProductos	NivelAbastecimiento	Tipo
PK, SA	NN, CK (MAYOR A CERO)	NN, CK (MAYOR A CERO)	FK PRODUCTO.ID	FK SUCURSAL.ID	NN, CK (numero mayor o igual)	NN, CK (numero mayor a cero)	NN, CK (MAYOR A CERO)	NN, CK ('Bodega' O 'Estante')

Figura 4. Tabla ALMACENAMIENTO, modelo relacional.

TABLA TRANSACCION					
idProducto	Cantidad	NumeroFactura	Costo	Promocion	Estado
FK PRODUCTO.ID	NN, CK (MAYOR A CERO)	FK FACTURA.NUMERO	NN, CK (MAYOR A CERO)	FK PROMOCION.ID	CK IN('pendiente', 'pagada')

Figura 5. Tabla TRANSACCIÓN, modelo relacional.

TABLA PROMOCION						
ID	Producto	Precio	Descripcion	FechaInicio	FechaFin	UnidadesDisponibles
PK,SA	FK PRODUCTO.ID	NN, CK (MAYOR A CERO)	NN	NN	NN, CK (debe ser igual o posterior a la fecha de inicio)	NN, CK (numero mayor o igual a cero)

Figura 6. Tabla PROMOCIÓN, modelo relacional.

En cuanto al modelo relacional, fue necesario agregar las tablas CARRITO y CONTIENE para ajustarse al modelo conceptual UML modificado. Las restricciones y llaves primarias y foráneas fueron agregadas conforme al modelo. Además, se creó la tabla ALMACENAMIENTO en respuesta a la modificación de Bodega y Estante y se modificaron las tablas TRANSACCIÓN y PROMOCIÓN.

3 Resultados

3.1 Resultados logrados

Modelo conceptual y relacional:

Al probar los modelos en el desarrollo de la aplicación, nos dimos cuenta que con las modificaciones de las tablas en *SQL Developer* de acuerdo a los nuevos requerimientos, el modelo se ajustaba y permanecía correcto, las nuevas tablas se crearon con sus especificaciones y chequeos pertinentes (CARRITO, CONTIENE, ALMACENAMIENTO). Las tablas permitieron realizar consultas para cumplir con los nuevos requerimientos funcionales y de consulta de la aplicación SuperAndes.

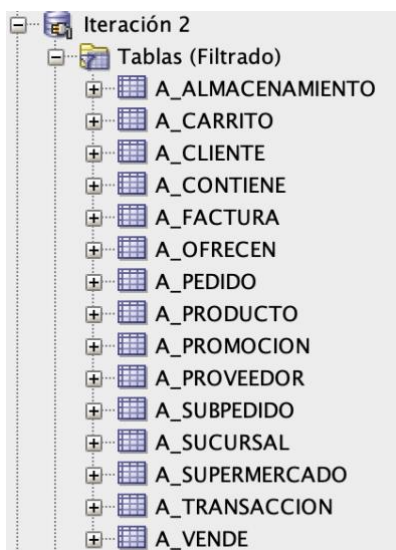


Figura 7. Tablas actualizadas en *SQLDeveloper*.

```

ALTER TABLE a_carrito
ADD CONSTRAINT ck_carrito_estado CHECK ( estado IN (
    'en uso',
    'abandonado',
    'pagado',
    'libre'
) ) ENABLE;

ALTER TABLE a_carrito
ADD CONSTRAINT ck_car_clave CHECK ( clave > -1 ) ENABLE;

ALTER TABLE a_carrito
ADD CONSTRAINT fk_c_sucursal FOREIGN KEY ( sucursal )
REFERENCES a_sucursal ( id )
ENABLE;

-- Creaci3n de la tabla ofrecen y especificaci3n de sus restricciones.

CREATE TABLE a_contiene (
    producto NUMBER,
    carrito NUMBER NOT NULL,
    cantidad NUMBER NOT NULL,
    CONSTRAINT a_contiene_pk PRIMARY KEY ( producto,
                                           carrito,
                                           cantidad )
);

ALTER TABLE a_contiene
ADD CONSTRAINT fk_contiene_producto FOREIGN KEY ( producto )
REFERENCES a_producto ( id )
ENABLE;

ALTER TABLE a_contiene
ADD CONSTRAINT fk_c_carrito FOREIGN KEY ( carrito )
REFERENCES a_carrito ( id )
ENABLE;

ALTER TABLE a_contiene ADD CONSTRAINT ck_contiene_cantidad CHECK ( cantidad > 0 ) ENABLE;

-- Creaci3n de la tabla promoci3n y especificaci3n de sus restricciones.

```

Figura 8. Actualizaci3n del documento de Esquema SuperAndes con las nuevas tablas agregadas.

Desarrollo de nuevos requerimientos:

En cuanto al desarrollo de los nuevos requerimientos, fue necesario implementar las clases de negocio y persistencia en el proyecto de Java. De esta manera las clases de Carrito, VOCarrito; Contiene, VOContiene y Almacenamiento, VOAlmacenamiento fueron creadas, modelando el negocio de SuperAndes con los atributos b3sicos. As3 mismo, se crearon las clases de persistencia SQLCarrito, SQLContiene y SQLAlmacenamiento, vincul3ndolas a la clase principal de persistencia.

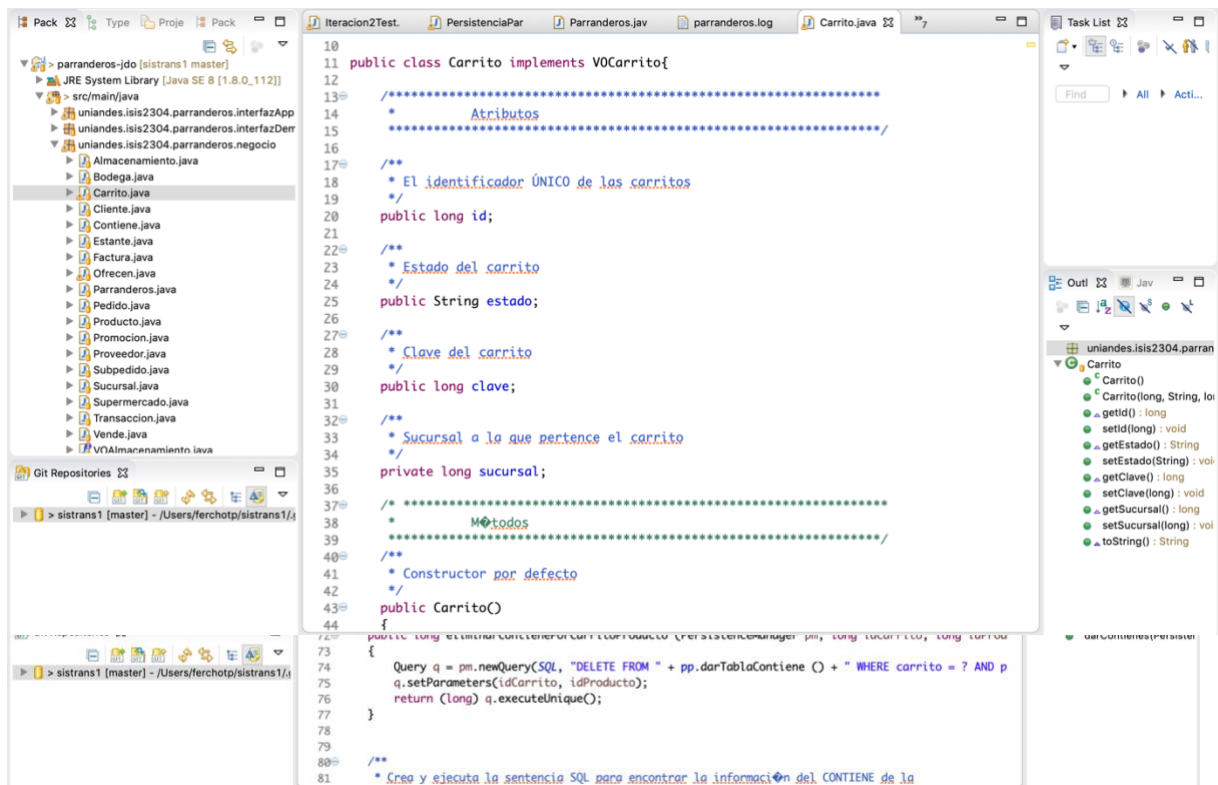


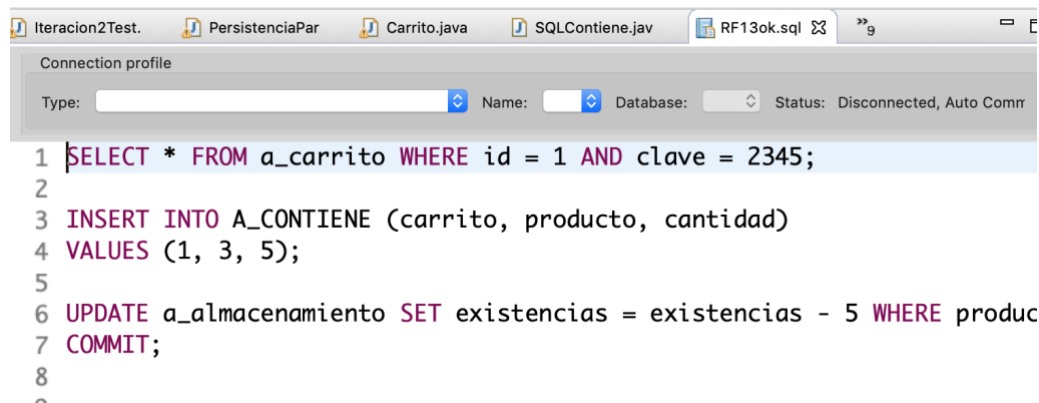
Figura 10. Actualizaci3n del proyecto Java del paquete de persistencia de SuperAndes.

Como resultado de la actualización, las clases se ajustaron adecuadamente al modelo y no ocasionaron ningún conflicto en las relaciones.

Sentencias SQL para requerimientos:

En esta parte, realizamos completamente los requerimientos funcionales RF12 al RF17, al ser requerimientos de una complejidad media alta, la sentencia SQL por sí sola no los podía resolver, por esta razón fueron manejados en Java para que cumplieran con el propósito de los requerimientos. En un caso por resaltar, el manejo del carrito fue modela de la siguiente manera:

Un carrito tiene un identificador y una clave. La clave la ingresa el mismo cliente que solicita el carrito y de esta manera el cliente se convierte en el dueño del mismo. El carrito tiene tres estados: libre, en uso y abandonado, de esta manera se pudieron manejar los diferentes requerimientos en donde el estado del carrito cambiaba dependiendo de la acción realizada.



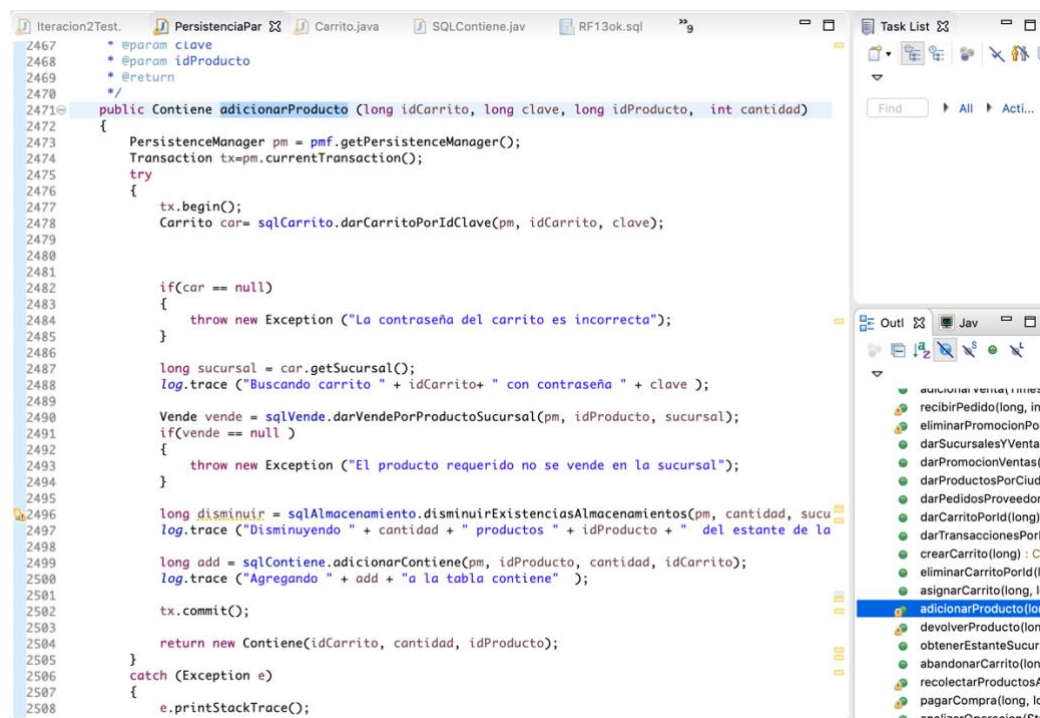
The screenshot shows a SQL editor window with a connection profile at the top. Below it, several SQL statements are listed, numbered 1 through 8. The statements are as follows:

```

1 SELECT * FROM a_carrito WHERE id = 1 AND clave = 2345;
2
3 INSERT INTO A_CONTIENE (carrito, producto, cantidad)
4 VALUES (1, 3, 5);
5
6 UPDATE a_almacenamiento SET existencias = existencias - 5 WHERE produc
7 COMMIT;
8

```

Figura 11. Sentencia SQL para el RF13.



The screenshot shows a Java IDE with the implementation of the RF13 requirement in the Contiene class. The code is as follows:

```

2467 * @param clave
2468 * @param idProducto
2469 * @return
2470 */
2471 public Contiene adicionarProducto (long idCarrito, long clave, long idProducto, int cantidad)
2472 {
2473     PersistenceManager pm = pmf.getPersistenceManager();
2474     Transaction tx=pm.currentTransaction();
2475     try
2476     {
2477         tx.begin();
2478         Carrito car= sqlCarrito.darCarritoPorIdClave(pm, idCarrito, clave);
2479
2480
2481         if(car == null)
2482         {
2483             throw new Exception ("La contraseña del carrito es incorrecta");
2484         }
2485
2486         long sucursal = car.getSucursal();
2487         log.trace ("Buscando carrito " + idCarrito+ " con contraseña " + clave );
2488
2489         Vende vende = sqlVende.darVendePorProductoSucursal(pm, idProducto, sucursal);
2490         if(vende == null )
2491         {
2492             throw new Exception ("El producto requerido no se vende en la sucursal");
2493         }
2494
2495         long disminuir = sqlAlmacenamiento.disminuirExistenciasAlmacenamientos(pm, cantidad, sucursal, idProducto);
2496         log.trace ("Disminuyendo " + cantidad + " productos " + idProducto + " del estante de la sucursal");
2497
2498         long add = sqlContiene.adicionarContiene(pm, idProducto, cantidad, idCarrito);
2499         log.trace ("Agregando " + add + " a la tabla contiene ");
2500
2501         tx.commit();
2502
2503         return new Contiene(idCarrito, cantidad, idProducto);
2504     }
2505     catch (Exception e)
2506     {
2507         e.printStackTrace();
2508     }
2509 }

```

Figura 12. Método en java que implementa el RF13.

Como podemos ver en las imágenes, la sentencia SQL para el RF13 sirvió de modelo para el manejo del requerimiento en *Java*, sirvió como idea del proceso que se debe de llevar a cabo para cumplir con el requerimiento. Por otro lado, en *Java* el requerimiento se desarrolla en su totalidad teniendo en cuenta las restricciones necesarias y valores a tener en cuenta que el usuario ingresa y que ya son fijos.

Interfaz con requerimientos integrados:

En la interfaz que ya teníamos implementada, se agregaron los nuevos requerimientos de consulta y funcionales, teniendo en cuenta que en la clase de interfaz se manejaba el método de la persistencia para obtener los valores de respuesta correctos.

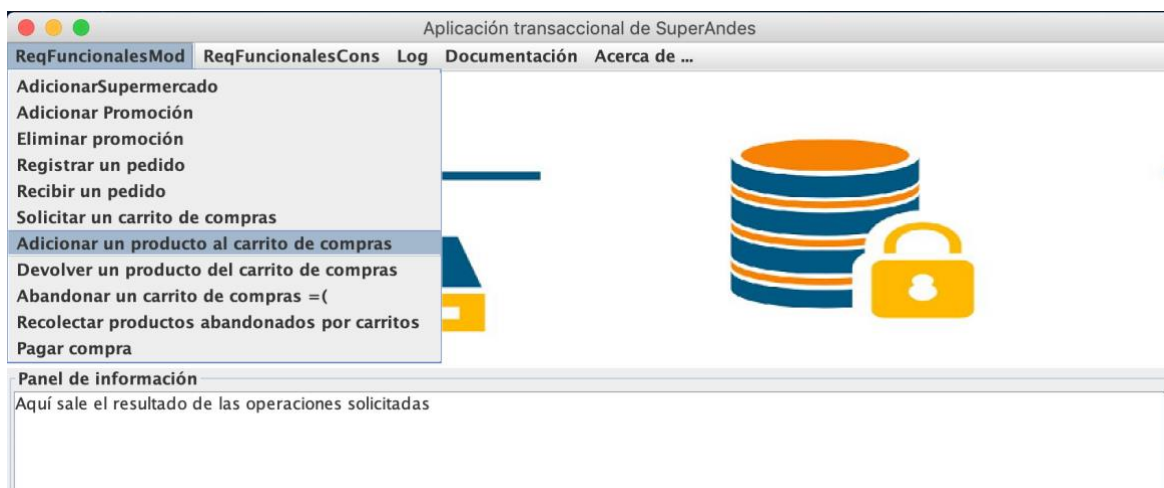
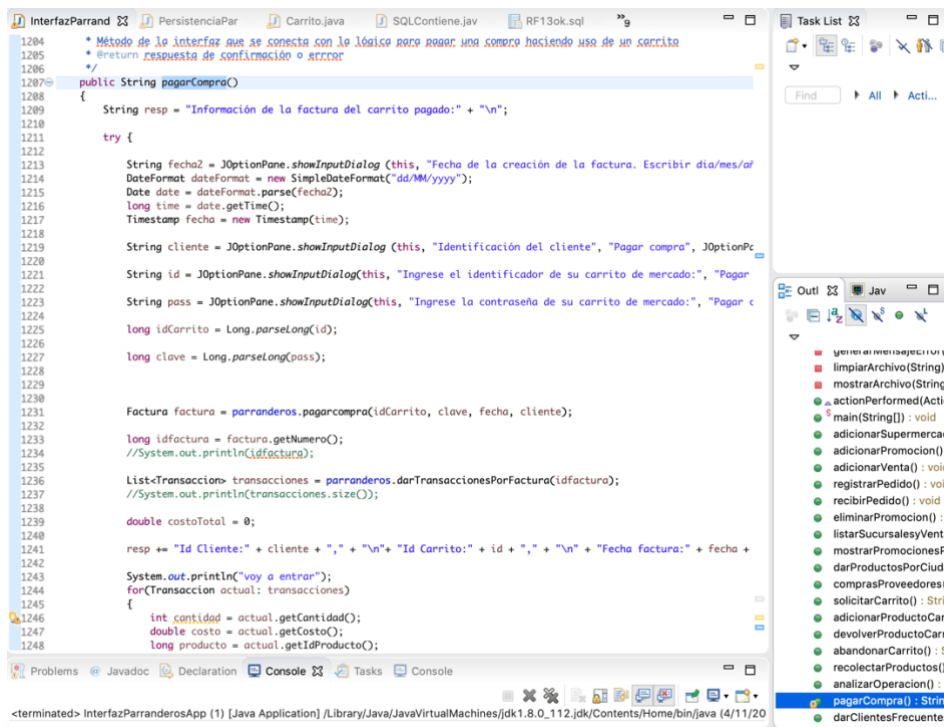


Figura 13. Interfaz de SuperAndes con los nuevos requerimientos agregados.



La interfaz y el log de datanucleus y superandes fue de gran ayuda para poder darnos cuenta de los errores que teníamos en los requerimientos y para poder desplegar la información de la manera más adecuada, a continuación, podemos ver uno de los resultados de los requerimientos de consulta (RFC7) en a interfaz de la aplicación.

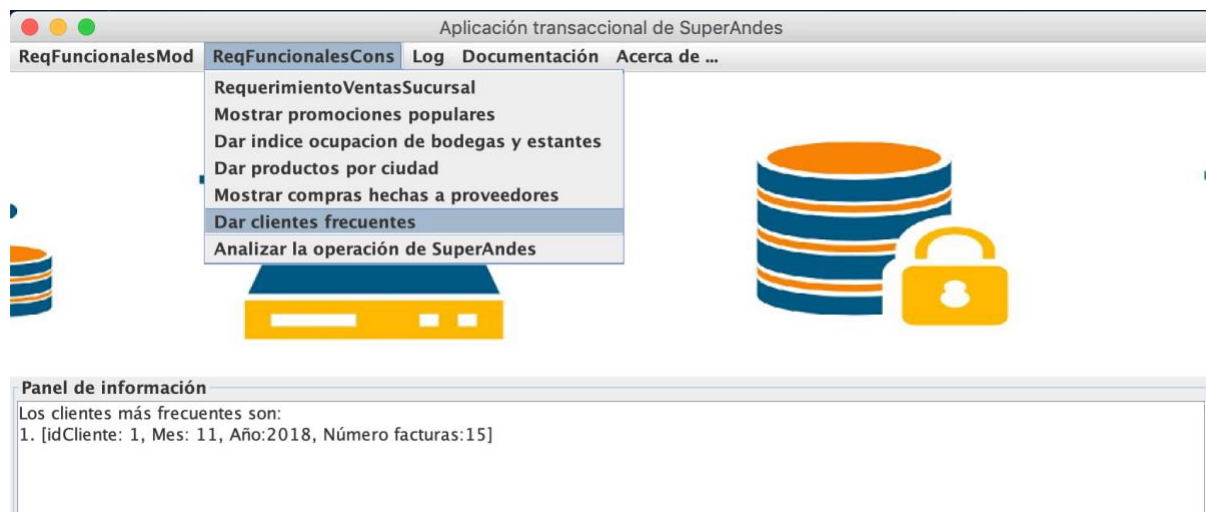


Figura 15. Interfaz con el resultado del requerimiento de consulta 7 de clientes más frecuentes.

3.2 Resultados no logrados

En cuanto a resultados no logrados, se destacan las pruebas, pues en algunas de estas no fue posible obtener resultados favorables para el negocio. Otro inconveniente que existió, fueron los requerimientos funcionales de consulta, donde específicamente el 7 no obtuvo los resultados esperados, pues no se logró hallar una manera de poder analizar la operación de SuperAndes de acuerdo a semanas, entonces en nuestro método solo se puede analizar por mes.

4 Balance plan de pruebas

Para la realización de las pruebas unitarias para las tablas, se realizó una clase de *Tests* llamada *Iteracion2Test*, que contenía tests para todos los requerimientos funcionales de la aplicación.

El funcionamiento de la clase un método de test por cada requerimiento. El contenido del test prueba reglas de funcionamiento básico del mismo, como por ejemplo, comprobación de inserciones y eliminaciones, verificación de actualización de estantes, comprobación de asignación de carrito correcto y demás. Todas estas con el fin de probar si las transacciones modeladas funcionan o no.

En cuanto a los resultados de pruebas, de las 7 pruebas ejecutadas, todas corrieron y 4 dieron resultados correctos sin errores y 2 con fallas por temas de asserts erróneos. (72% de efectividad). De igual manera como en la anterior iteración, creemos que por las operaciones lógicas de los métodos requerimientos, que son de cierta manera robustas, al correr los test encontrábamos con que *Eclipse* se en un loop infinito hasta que salía la excepción de *StackOverflow* por lo que creemos que en algunos casos la memoria virtual en *Eclipse* se llenaba.

5 Análisis reglas de negocios

Una vez finalizado el proyecto, se encontraron diferentes cambios y análisis de las reglas de negocio de la aplicación.

Por un lado con la implementación del nuevo requerimiento de Carrito de compras, se establecieron nuevas reglas de negocio para SuperAndes. Es de suma importancia tener en cuenta que el carrito le pertenece a solo un cliente a la vez. Así mismo, el carrito puede no tener un "dueño" y estar en estado libre o abandonado por lo que esa regla de negocio se debe asegurar y modelar adecuadamente. En este caso logramos esto con el uso de una clave para cada carrito. La idea consiste en que se tienen 20 carritos en superandes y todos inician con clave en 0 y en estado libre, el cliente al solicitar un carrito, define la clave para este y automáticamente se le asigna en orden el primer carrito libre en la base de datos. Cuando el cliente deja ir el carrito o paga la compra, la clave del carrito se reinicia a 0 y su estado cambia ya sea a libre o abandonado de haber estado en uso.

En cuanto a los requerimientos de manejo del carrito, como adicionar productos, devolver, abandonar carrito, recolectar productos y pagar compra hay una regla sumamente importante que debe asegurarse en todo momento para mantener los datos consistentes y coherentes con el mundo de negocio. Este aspecto es el del manejo del almacenamiento de estantes y bodegas en el supermercado. Por cada uno de los requerimientos mencionados del carrito, se debe tener en cuenta cuando disminuir existencias o cuando aumentarlas de acuerdo a los productos que se manejen en el carrito, es muy importante que la base de datos se mantenga consistente y que si un producto se agrega al carrito, que sea eliminado de las existencias en almacenamiento. Así mismo se deben tener en cuenta el nivel de abastecimiento y reorden con el fin de realizar pedidos a proveedores si es necesario. Este aspecto mantendrá la consistencia de los datos en los procesos de manejo del carrito de compras.

Como regla de negocio general, se debe asegurar que se cumplan los aspectos ACID para el manejo de información en una aplicación transaccional. Este aspecto específicamente a la Atomicidad, la Consistencia, el Aislamiento y la Durabilidad de la información. Sin estos aspectos la aplicación no cumpliría con los aspectos cruciales de una aplicación de este estilo y por eso tendría muchos problemas con el manejo de datos. Esta regla de negocio se manejó tanto en *SQLDeveloper* como en *Java*, teniendo en cuenta los niveles de aislamiento y el *autocommit*. Así mismo, en la lógica se asegura y se verifican en todos los requerimientos, que los datos sean consistentes, que no se repitan y que se cumplan las reglas ACID.

6 Conclusiones

- Integramos requerimientos funcionales y no funcionales relacionados con los aspectos ACID de una aplicación transaccional desarrollada en una arquitectura de tres niveles con manejo de persistencia en base de datos con el fin de completar el desarrollo de la aplicación SuperAndes.
- Fuimos capaces de modificar el modelo conceptual y relacional conforme a los nuevos requerimientos funcionales y no funcionales establecidos para esta iteración, de esta manera logramos llevar a cabo el desarrollo de la aplicación y aplicar conceptos aprendidos en clase.
- Incorporamos elementos de calidad del modelo de datos, con respecto a integridad de la información, la transaccionalidad y el aislamiento de datos. Esto consistió en un buen

análisis del caso de negocio para poder así implementar el proyecto de la mejor manera de tal forma que la información sea íntegra, persistente y consistente y se ciñera a los aspectos básicos ACID. Ya que esta es la única forma en que se puede manejar un caso de negocio que maneja procesos transaccionales.

- Implementamos una aplicación informática de mediana complejidad (SuperAndes) con requerimientos nuevos que supusieron un reto de desarrollo y que contiene bases de datos relacionales, siguiendo la arquitectura de referencia definida para el curso. Aprendimos a utilizar las diferentes herramientas de modelaje (GenMyModel) e implementación de bases de datos (SQL Developer) para poder así desarrollar la aplicación con todas sus funciones.
- Aprendimos a identificar y desarrollar los diferentes y nuevos requerimientos funcionales y de consulta de una aplicación como lo fue SuperAndes. Todo esto teniendo en cuenta las reglas de negocio y las diferentes condiciones para que cumpliera los aspectos ACID.
- Aprendimos del trabajo realizado en la anterior iteración y dividimos el trabajo de manera organizada para poder sacar adelante el proyecto. Es de suma importancia organizar el tiempo y las tareas a realizar por cada integrante del grupo, ya que en muchas ocasiones genera que las contribuciones no sean equitativas y que el tiempo no sea suficiente para acabar el desarrollo del proyecto.
- Es de gran enseñanza conocer el uso correcto de las bases de datos para plantear soluciones informáticas pertinentes y funcionales. El manejo de información es de gran importancia en los grandes negocios y es por esto que su buen uso y conocimiento permitirá mantener los datos a través del tiempo y de manera íntegra.
- Entendimos más a fondo el manejo de la información, de la importancia del cumplimiento de las reglas ACID, ya que estas definen una buena base de datos y permiten que los datos siempre sean durables y coherentes con la realidad.

7 Bibliografía

- GARCIA-MOLINA, Hector, ULLMAN, Jeffrey, WIDOM, Jennifer. "Database Systems: The complete Book". 2nd. Edition. Prentice Hall, 2009.
- CHURCHER, Claire. "Beginning Database Design. From novice to professional". APress. 2007.