“Proyecto del curso: SuperAndes – Iteración 1”

Juan F. Torres Piza, Nicolás Cobos Carranza

El propósito de este documento es recopilar el análisis y resultados obtenidos a través del desarrollo del modelaje e implementación de la aplicación de un supermercado, SuperAndes. Este documento muestra paso a paso el desarrollo de la primera iteración del proyecto desde los modelos diseñados hasta la implementación de la base de datos, de la aplicación y de sus pruebas respectivas.

Universidad de los Andes, Bogotá, Colombia

{jf.torresp, n.cobos}@uniandes.edu.co

Fecha de presentación: Octubre 3 de 2018

Tabla de contenido

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 1 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 planteamiento del modelo conceptual, hasta el balance de pruebas y análisis de las reglas de negocio. El documento se divide en 5 partes: la primera ilustra los modelos conceptuales y relacionales obtenidos con el análisis del enunciado de SuperAndes, la segunda parte muestra los resultados obtenidos en el desarrollo de toda la iteración, esto incluye los requerimientos funcionales, no funcionales y de consulta, desarrollados en *SQL Developer* y en *Eclipse*. La tercera parte muestra un reporte de resultados de las pruebas de unicidad e integridad de las tablas. Finalmente se mostrará el análisis de las reglas de negocio sobre cuáles se cumplieron, cuáles no y cuáles fue necesario agregar.

2 Análisis del caso

Luego de haber leído y analizado el caso entregado, se pudo establecer que los elementos fundamentales que hacen parte del negocio que se describe son:

- Roles de usuario: teniendo en cuenta que los roles se refieren a la colección de permisos que poseen los usuarios, existen dos tipos de usuario, de acuerdo a sus roles. Existe el administrador, que es capaz de visualizar y modificar la información de todos los supermercados, las sucursales y en general toda la información del negocio. Por otra parte, existen los supermercados, que poseen menos permisos pues solo pueden visualizar y modificar la información que este asociada a ellos. En este caso, la aplicación está desde el punto de vista del administrador.
- Entidades de negocio: identificamos 11 entidades de negocio como clases en UML y 3 clases de asociación. Las más importantes son supermercado, sucursal, producto, factura, proveedor, cliente y pedido.
- Funcionalidades principales: los requerimientos funcionales se especifican en un documento anexo, pero las funcionalidades principales que la aplicación debería tener se asocian a la búsqueda específica de información útil para la generación de reportes de rendimiento y a la creación de entidades de negocio vitales para el usuario (ejemplo: pedido de productos a proveedores o creación de promociones para aumentar ventas).
- Reglas de negocio: existen muchas reglas de negocio por lo que sería poco práctico listarlas acá, entonces solo se mencionarán las principales: cada vez que exista un déficit para poder suplir la demanda de productos por parte de clientes se genera un pedido de los faltantes a un proveedor, cada vez que se haga recepción de un pedido se tiene que verificar que exista la capacidad de almacenar los productos, los precios de los productos ofrecidos pueden variar de acuerdo a la sucursal, y las sucursales pueden calificar el servicio de los proveedores luego de haber recibido un pedido de productos.

3 Modelos

3.1 Modelo Conceptual

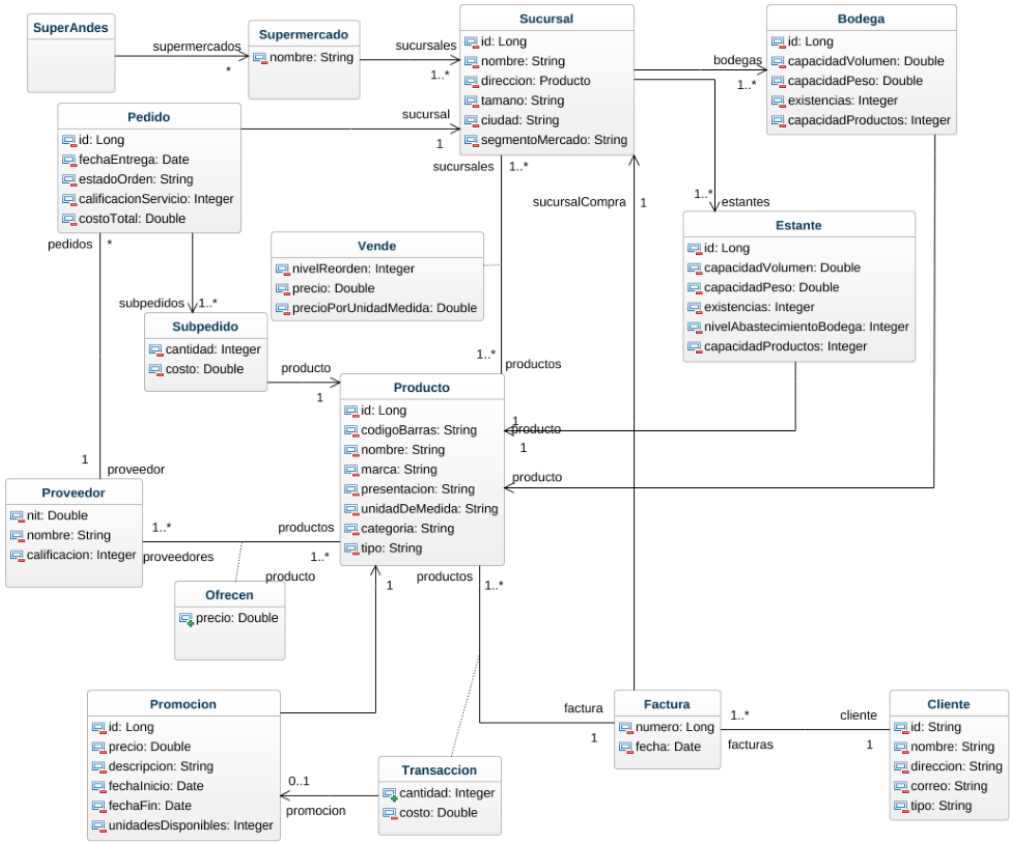


Figura 1. Modelo UML generado en GenMyModel

El modelo conceptual de SuperAndes cuenta con 14 clases que tratan de reunir todo el funcionamiento un supermercado, desde sus sucursales hasta el proceso de transacción de un producto por parte de un cliente. Hay diferentes clases de asociación en el modelo como los productos OFRECIDOS por un proveedor, la TRANSACCIÓN de la compra de un producto y la VENTA de productos por parte de una sucursal. En el caso de TRANSACCIÓN esta se implementó con el fin de que en una factura hubiese varias transacciones y que cada transacción estuviese asociada a un producto, así podíamos tener una factura con diferentes productos. Las clases que deben ser persistentes, en este caso, son todas puesto que para los requerimientos funcionales se deben hacer asociaciones entre las diferentes tablas y entidades para poder sacar la información que se requiere.

3.2 Modelo Relacional

Este es el modelo con tablas que realizamos de acuerdo a nuestro modelo UML:

TABLA SUPERMERCADO
Nombre
PK,UA
Bogotá

Figura 2. Tabla SUPERMERCADO, modelo relacional.

TABLA SUCURSAL						
ID	Ciudad	Dirección	SegmentoMercado	Tamano	Nombre	Supermercado
PK, SA	NN	NN	NN	NN, CK(NUMERO POSITIVO)	NN	FK SUPERMERCADO.NOMBRE

Figura 3. Tabla SUCURSAL, modelo relacional.

TABLA BODEGA					
ID	CapacidadVolumen	CapacidadPeso	Producto	Sucursal	Existencias
PK, SA	NN, CK (MAYOR A CERO)	NN, CK (MAYOR A CERO)	FK PRODUCTO.ID	FK SUCURSAL.ID	NN, CK (numero mayor o igual a cero)

Figura 4. Tabla BODEGA, modelo relacional.

TABLA ESTANTE						
ID	CapacidadVolumen	CapacidadPeso	NivelAbastecimientoBodega	Existencias	Producto	Sucursal
PK	NN, CK (MAYOR A CERO)	NN, CK (MAYOR A CERO)	NN, CK (MAYOR A CERO)	NN, CK (MAYOR A CERO)	FK PRODUCTO.ID	FK SUCURSAL.ID

Figura 5. Tabla ESTANTE, modelo relacional.

TABLA PRODUCTO							
ID	Nombre	Marca	Presentacion	UnidadDeMedida	CodigoBarras	Categoria	Tipo
PK, SA	NN	NN	NN	NN	NN, CK (Hexadecimal)	NN	NN

Figura 6. Tabla PRODUCTO, modelo relacional.

TABLA VENDE				
Sucursal	IdProducto	NivelReorden	PrecioUnitario	PrecioPorUnidadMedida
FK (SUCURSAL.ID)	FK (PRODUCTO.ID)	NN, CK (MAYOR A CERO)	NN, CK (MAYOR A CERO)	NN, CK (MAYOR A CERO)

Figura 7. Tabla VENDE, modelo relacional.

TABLA PROVEEDOR		
NIT	Nombre	Calificacion
PK,UA	NN	NN, CK (MAYOR A CERO, MENOR O IGUAL A DIEZ)

Figura 8. Tabla PROVEEDOR, modelo relacional.

TABLA PEDIDO						
ID	Sucursal	Proveedor	FechaEntrega	EstadoOrden	CalificacionServicio	CostoTotal
PK, SA	FK(SUCURSAL.ID)	FK (PROVEEDOR.ID)	NN, CK (NO PUEDE SER UNA FECHA ANTERIOR A LA ACTUAL)	NN, CK ('PENDIENTE', 'ENTREGADO')	NN, CK (MAYOR A CERO, MENOR O IGUAL A DIEZ)	NN, CK (MAYOR A CERO)

Figura 9. Tabla PEDIDO, modelo relacional.

TABLA SUBPEDIDO			
IDPedido	Producto	Cantidad	Costo
NN, FK (PEDIDO.ID)	NN, FK(PRODUCTO.ID)	NN, CK (mayor a cero)	NN, CK (mayor a cero)

Figura 10. Tabla SUBPEDIDO, modelo relacional.

TABLA OFRECEN		
Producto	Proveedor	Costo
FK PRODUCTO.ID	FK PROVEEDOR.NIT	NN, CK (MAYOR A CERO)

Figura 11. Tabla OFRECEN, modelo relacional.

TABLA FACTURA			
Numero	Fecha	Cliente	Sucursal
PK,SA	NN, CK (NO PUEDE SER UNA FECHA POSTERIOR A LA ACTUAL)	FK CLIENTE_PERSONA.ID	FK SUCURSAL.ID

Figura 12. Tabla FACTURA, modelo relacional.

TABLA TRANSACCION				
Producto	Cantidad	NumeroFactura	Costo	Promocion
FK PRODUCTO.ID	NN, CK (MAYOR A CERO)	FK FACTURA.NUMERO	NN, CK (MAYOR A CERO)	FK PROMOCION.ID

Figura 13. Tabla TRANSACCION, 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 14. Tabla PROMOCION, modelo relacional.

TABLA CLIENTE				
ID	Nombre	Direccion	Correo	Tipo
PK,UA	NN	NN	NN, UNIQUE	NN CK('PERSONA', 'EMPRESA')

Figura 15. Tabla CLIENTE, modelo relacional.

Paralelamente, empleando la herramienta Enterprise Architect, generamos un posible modelo relacional (DDL) de acuerdo al UML anteriormente presentado. Debido al tamaño de este modelo, preferimos no adjuntarlo debido a que podría ser difícil de interpretar por partes, sin embargo, el modelo está adjuntado en el archivo del proyecto. El modelo no está en la forma de tablas sino en este estilo:

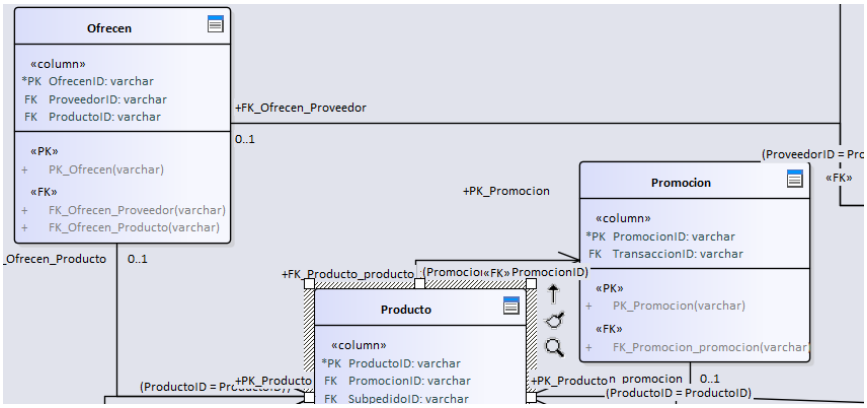


Figura 16. Muestra del modelo generado en Enterprise Architect.

Con respecto a este, honestamente fue difícil de analizar, pero pudimos establecer que algunas de las llaves foráneas que se exponen en el modelo de *Enterprise Architect*, también las pusimos en nuestro modelo de tablas. También existen algunas inconsistencias en esas mismas llaves pues, por ejemplo, la clase `Subpedido` debería tener una llave foránea de producto, sin embargo, en el modelo de *Enterprise Architect* esto se muestra al revés. Es por eso que, siguiendo el consejo de nuestros profesores de Sistemas Transaccionales, estas herramientas son una gran ayuda para el modelado de casos, pero no se les debería confiar plenamente.

4 Resultados

4.1 Resultados logrados

Modelo Conceptual y Relacional:

Al probar los modelos en el desarrollo de la aplicación, nos dimos cuenta que con la creación de las tablas en *SQL Developer*, el modelo se ajustaba y era correcto, las tablas se crearon con sus especificaciones y chequeos pertinentes. Las tablas permitieron realizar consultas para cumplir con los requerimientos funcionales de la aplicación SuperAndes.

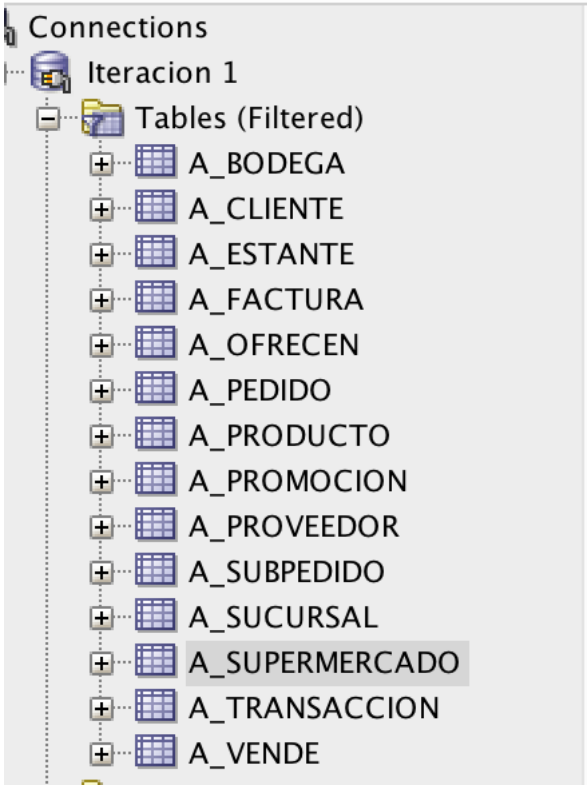


Figura 17. Creación Tablas en SQL Developer.

```
-- Copie el contenido de este archivo en una pestaÛa SQL de SQL Developer
-- Ejec˙telo como un script - Utilice el botÛn correspondiente de la pestaÒa utilizada

-- CreaciÛn del secuenciador
create sequence superandes_sequence;

-- CreaciÛn de la tabla supermercado y especificaciÛn de sus restricciones.
CREATE TABLE A_SUPERMERCADO
   (NOMBRE VARCHAR2(255 BYTE),
   CONSTRAINT A_SUPERMERCADO_PK PRIMARY KEY (NOMBRE));

-- CreaciÛn de la tabla SUCURSAL y especificaciÛn de sus restricciones
CREATE TABLE A_SUCURSAL
   (ID NUMBER,
    NOMBRE VARCHAR2(255 BYTE) NOT NULL,
    CIUDAD VARCHAR2(255 BYTE) NOT NULL,
    DIRECCION VARCHAR2(255 BYTE) NOT NULL,
    SEGMENTOMERCADO VARCHAR2(255 BYTE) NOT NULL,
    TAMANO NUMBER NOT NULL,
    SUPERMERCADO VARCHAR2(255 BYTE) NOT NULL,
    CONSTRAINT A_SUCURSAL_PK PRIMARY KEY (ID));

ALTER TABLE A_SUCURSAL
ADD CONSTRAINT fk_s_supermercado
    FOREIGN KEY (supermercado)
    REFERENCES a_supermercado(nombre)
ENABLE;

ALTER TABLE A_SUCURSAL
ADD CONSTRAINT CK_S_TAMANO
    CHECK (tamano > 0)
ENABLE;

-- CreaciÛn de la tabla producto y especificaciÛn de sus restricciones.
CREATE TABLE A_PRODUCTO
   (ID NUMBER,
    NOMBRE VARCHAR2(255 BYTE) NOT NULL,
    MARCA VARCHAR2(255 BYTE) NOT NULL,
    PRESENTACION VARCHAR2(255 BYTE) NOT NULL,
    CODIGOBARRAS VARCHAR2 (255 BYTE) NOT NULL,
    UNIDADMEDIDA VARCHAR2(255 BYTE) NOT NULL,
    CATEGORIA VARCHAR2(255 BYTE) NOT NULL,
```

Figura 18. Documento .sql con la creación de las tablas.

Desarrollo del proyecto SuperAndes:

Para esta parte de la Iteración, utilizamos como base el proyecto en *Eclipse* de parranderos-jdo. De ahí empezamos a construir la aplicación teniendo en cuenta los paquetes de interfaz, negocio y persistencia.

En cuanto las clases de Negocio, fue posible su implementación y fue acorde a los modelos planteados. Las clases del SuperAndes cumplían las reglas de negocio y las asociaciones modeladas fueron consistentes y pertinentes.

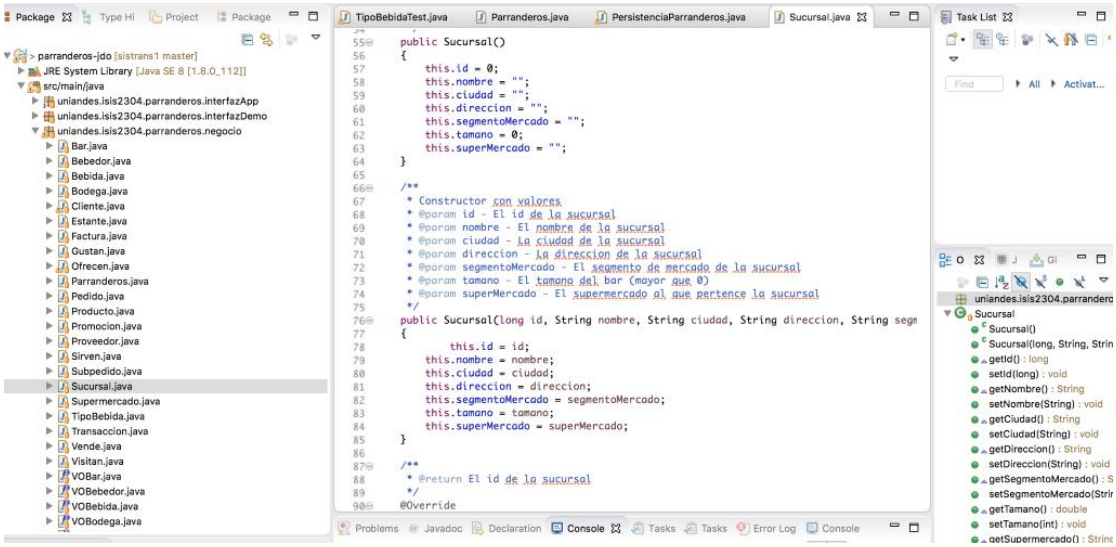


Figura 19. Captura de pantalla del proyecto en Eclipse con las clases de negocio.

Por otro lado, también tuvimos que implementar las clases de persistencia que conectaran el negocio con la base de datos para así poder realizar consultar y cumplir con los requerimientos funcionales de la aplicación. Este proceso se llevó a cabo exitosamente en el cual se implementaron las 14 clases de persistencia para SuperAndes, junto con sus métodos de consulta más importantes (ej. adicionar Promoción, eliminar promoción por id, dar promociones, etc.). Así, logramos integrar el negocio con sus funciones principales en la interfaz.

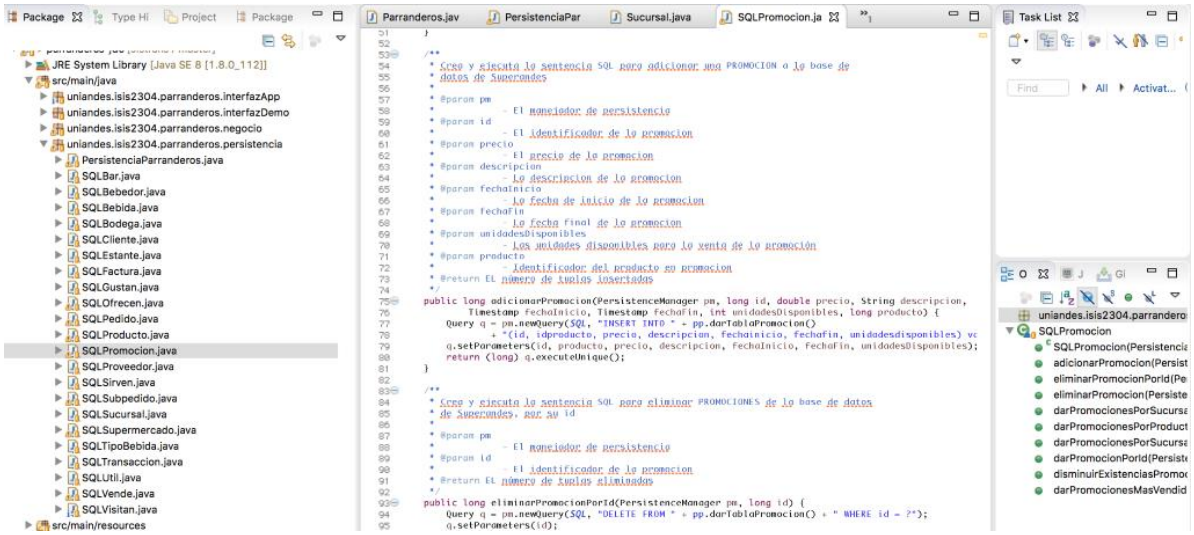
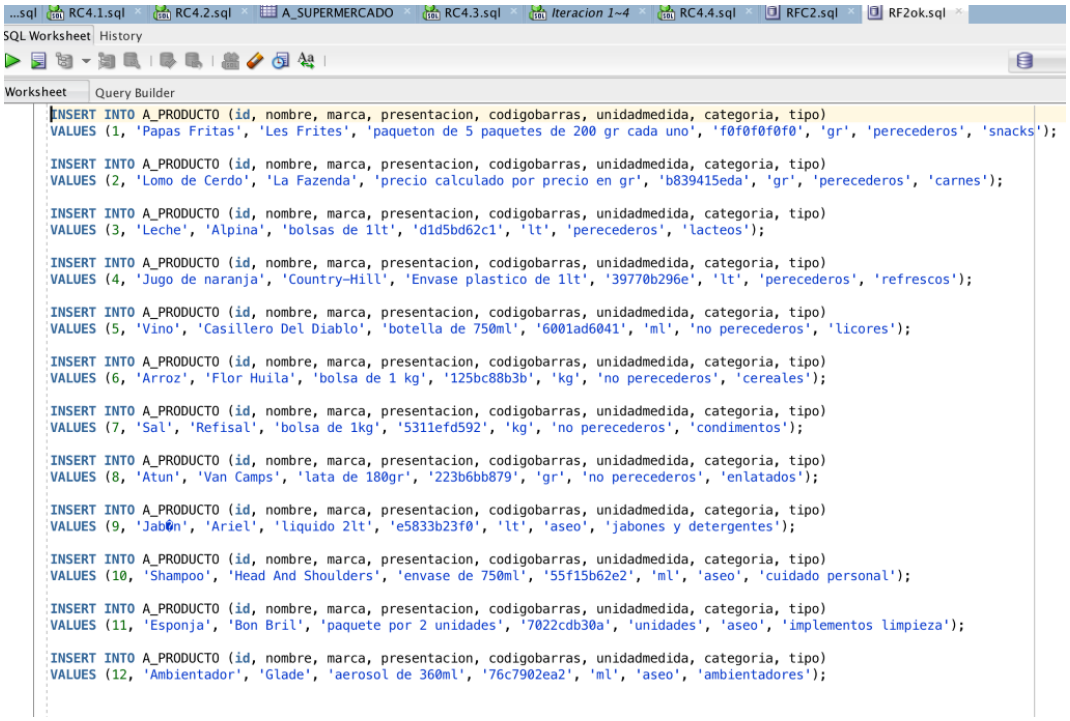


Figura 20. Captura de pantalla del proyecto en Eclipse con las clases de persistencia.

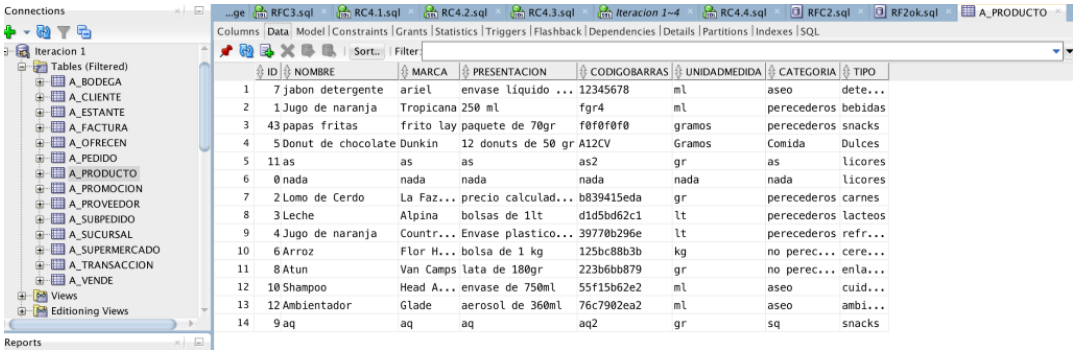
Sentencias SQL para Requerimientos:

En cuanto a los requerimientos, desarrollamos los archivos *.sql* para todos los requerimientos funcionales y los de consulta. Todos los archivos corrieron en SQL y cumplieron con lo propuesto en el enunciado.



```
INSERT INTO A_PRODUCTO (id, nombre, marca, presentacion, codigobarras, unidadmedida, categoria, tipo)
VALUES (1, 'Papas Fritas', 'Les Frites', 'paqueton de 5 paquetes de 200 gr cada uno', 'f0f0f0f0f0', 'gr', 'perecederos', 'snacks');

INSERT INTO A_PRODUCTO (id, nombre, marca, presentacion, codigobarras, unidadmedida, categoria, tipo)
VALUES (2, 'Lomo de Cerdo', 'La Fazenda', 'precio calculado por precio en gr', 'b839415eda', 'gr', 'perecederos', 'carnes');

INSERT INTO A_PRODUCTO (id, nombre, marca, presentacion, codigobarras, unidadmedida, categoria, tipo)
VALUES (3, 'Leche', 'Alpina', 'bolsas de 1lt', 'd1d5bd62c1', 'lt', 'perecederos', 'lacteos');

INSERT INTO A_PRODUCTO (id, nombre, marca, presentacion, codigobarras, unidadmedida, categoria, tipo)
VALUES (4, 'Jugo de naranja', 'Country-Hill', 'Envase plastico de 1lt', '39770b296e', 'lt', 'perecederos', 'refrescos');

INSERT INTO A_PRODUCTO (id, nombre, marca, presentacion, codigobarras, unidadmedida, categoria, tipo)
VALUES (5, 'Vino', 'Casillero Del Diablo', 'botella de 750ml', '6001ad6041', 'ml', 'no perecederos', 'licores');

INSERT INTO A_PRODUCTO (id, nombre, marca, presentacion, codigobarras, unidadmedida, categoria, tipo)
VALUES (6, 'Arroz', 'Flor Huila', 'bolsa de 1 kg', '125bc88b3b', 'kg', 'no perecederos', 'cereales');

INSERT INTO A_PRODUCTO (id, nombre, marca, presentacion, codigobarras, unidadmedida, categoria, tipo)
VALUES (7, 'Sal', 'Refisal', 'bolsa de 1kg', '5311efd592', 'kg', 'no perecederos', 'condimentos');

INSERT INTO A_PRODUCTO (id, nombre, marca, presentacion, codigobarras, unidadmedida, categoria, tipo)
VALUES (8, 'Atun', 'Van Camps', 'lata de 180gr', '223b6bb879', 'gr', 'no perecederos', 'enlatados');

INSERT INTO A_PRODUCTO (id, nombre, marca, presentacion, codigobarras, unidadmedida, categoria, tipo)
VALUES (9, 'Jab?n', 'Ariel', 'liquido 2lt', 'e5833b23f0', 'lt', 'aseo', 'jabones y detergentes');

INSERT INTO A_PRODUCTO (id, nombre, marca, presentacion, codigobarras, unidadmedida, categoria, tipo)
VALUES (10, 'Shampoo', 'Head And Shoulders', 'envase de 750ml', '55f15b62e2', 'ml', 'aseo', 'cuidado personal');

INSERT INTO A_PRODUCTO (id, nombre, marca, presentacion, codigobarras, unidadmedida, categoria, tipo)
VALUES (11, 'Esponja', 'Bon Bril', 'paquete por 2 unidades', '7022cdb30a', 'unidades', 'aseo', 'implementos limpieza');

INSERT INTO A_PRODUCTO (id, nombre, marca, presentacion, codigobarras, unidadmedida, categoria, tipo)
VALUES (12, 'Ambientador', 'Glade', 'aerosol de 360ml', '76c7902ea2', 'ml', 'aseo', 'ambientadores');
```

Figura 21. Captura de pantalla en SQL Developer con el requerimiento de agregar productos



	ID	NOMBRE	MARCA	PRESENTACION	CODIGOBARRAS	UNIDADMEDIDA	CATEGORIA	TIPO
1	7	jabon detergente	ariel	envase liquido ...	12345678	ml	aseo	dete...
2	1	Jugo de naranja	Tropicana	250 ml	fgr4	ml	perecederos	bebidas
3	43	papas fritas	frito lay	paquete de 70gr	f0f0f0f0	gramos	perecederos	snacks
4	5	Donut de chocolate	Dunkin	12 donuts de 50 gr	A12CV	Gramos	Comida	Dulces
5	11	as	as	as	as2	gr	as	licores
6	0	nada	nada	nada	nada	nada	nada	licores
7	2	Lomo de Cerdo	La Faz...	precio calculad...	b839415eda	gr	perecederos	carnes
8	3	Leche	Alpina	bolsas de 1lt	d1d5bd62c1	lt	perecederos	lacteos
9	4	Jugo de naranja	Countr...	Envase plastico...	39770b296e	lt	perecederos	refr...
10	6	Arroz	Flor H...	bolsa de 1 kg	125bc88b3b	kg	no perec...	cere...
11	8	Atun	Van Camps	lata de 180gr	223b6bb879	gr	no perec...	enla...
12	10	Shampoo	Head A...	envase de 750ml	55f15b62e2	ml	aseo	cuid...
13	12	Ambientador	Glade	aerosol de 360ml	76c7902ea2	ml	aseo	ambi...
14	9	aq	aq	aq	aq2	gr	sq	snacks

Figura 22. Captura de pantalla en SQL Developer con el resultado de la inserción de productos reales y de prueba.

En la figura 21 y 22 se pueden ver los resultados de uno de los requerimientos funcionales de SuperAndes, agregar productos al supermercado. En la figura 5 se puede ver la sentencia SQL que agrega diferentes productos con sus características correspondientes a la tabla de producto. Mientras que en la figura 22, se evidencian los resultados de la sentencia anterior.

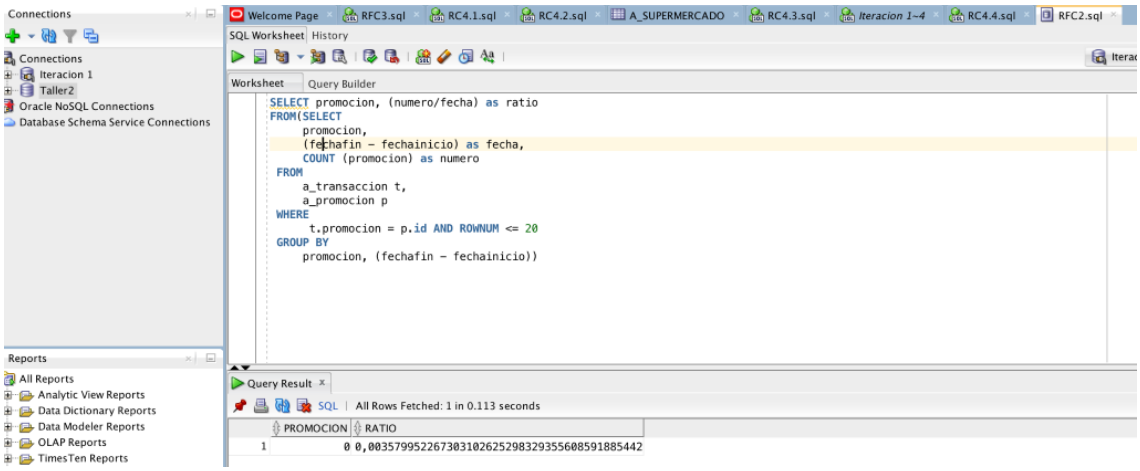


Figura 23. Captura de pantalla en SQL Developer con el RC2 funcionando.

En la Figura 5 se puede evidenciar la sentencia SQL que encuentra la promoción más popular que se vendió en el menor tiempo posible. Se puede ver en consola que da un resultado pertinente. Para este caso, la base de datos contaba con una sola promoción.

Interfaz con requerimientos integrados:

Para esta parte, se vinculó la interfaz con las clases de negocio y persistencia para facilitar al usuario el cumplimiento de los requerimientos funcionales y de consulta. Fue posible realizar esto ya que la mayoría de los requerimientos de consulta funcionaron en la aplicación y estaban conectados correctamente a la base de datos.

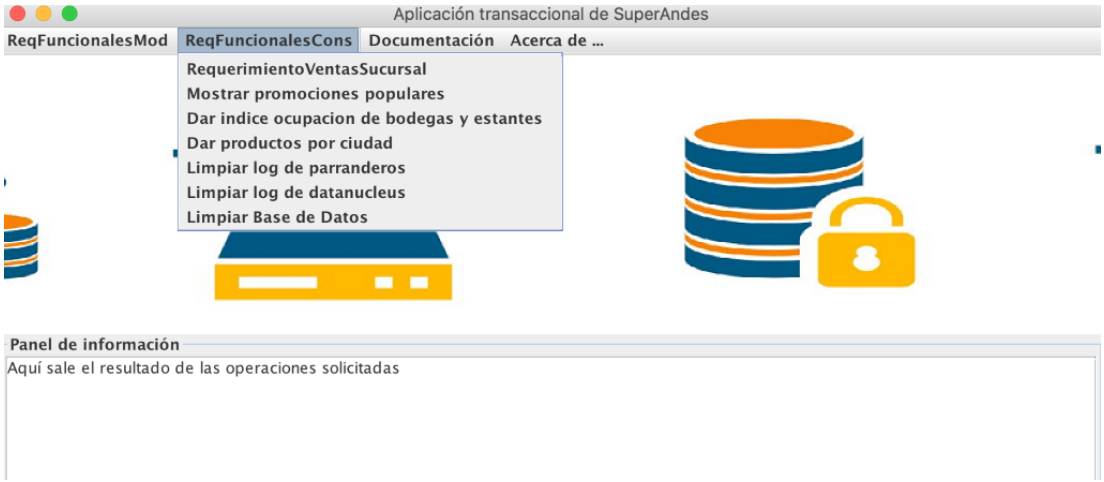


Figura 24. Captura de pantalla de la interfaz de SuperAndes con los req de consulta.

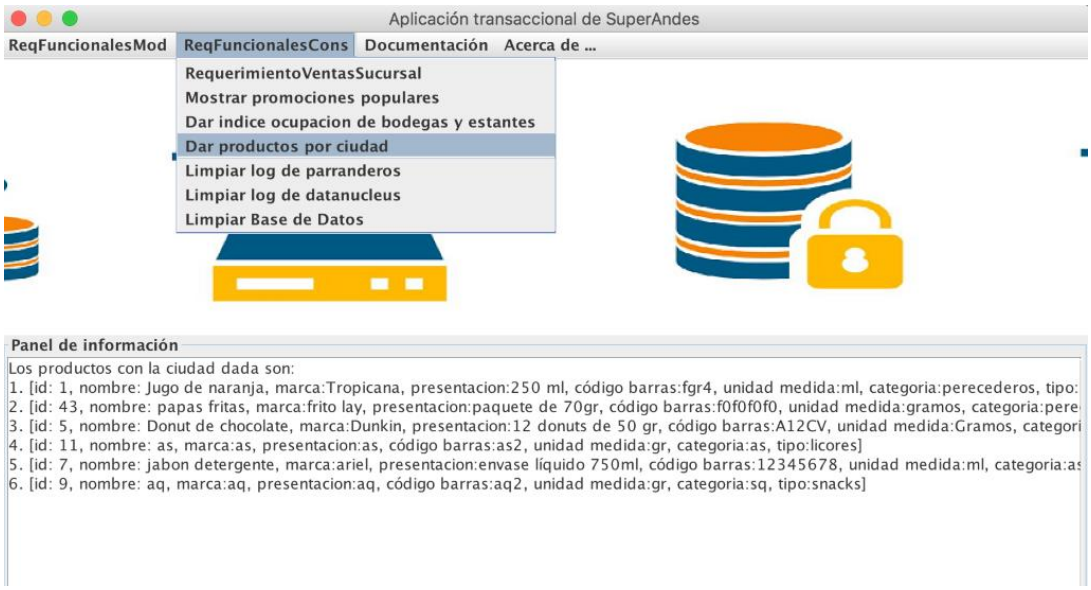


Figura 25. Captura de pantalla de la interfaz de SuperAndes con el funcionamiento del RC4.

En la figura 24 y 25 se ilustra la interfaz de SuperAndes con el correcto funcionamiento de uno de los Requerimientos de Consulta (RC4) donde el usuario puede obtener los productos dada una ciudad que él ingresa.

4.2 Resultados no logrados

Único proyecto Eclipse y conexión a base de datos SuperAndes:

Desafortunadamente, cuando creamos un repositorio aparte para el desarrollo del proyecto, nos basamos en el de parranderos-jdo pero este no corría y no se conectaba con la base de datos. Por esta razón, decidimos incluir las tablas de SuperAndes en la conexión a Parranderos y así mismo implementar todas las clases (negocio, persistencia e interfaz) en el proyecto de Eclipse de parranderos-jdo para que así estuviesen conectadas con la base de datos. Es por eso que existen clases y paquetes con el nombre de “parranderos”.

Figura 26. Captura de pantalla del error que impedía que se desplegara la interfaz.

Requerimientos de consulta en la interfaz:

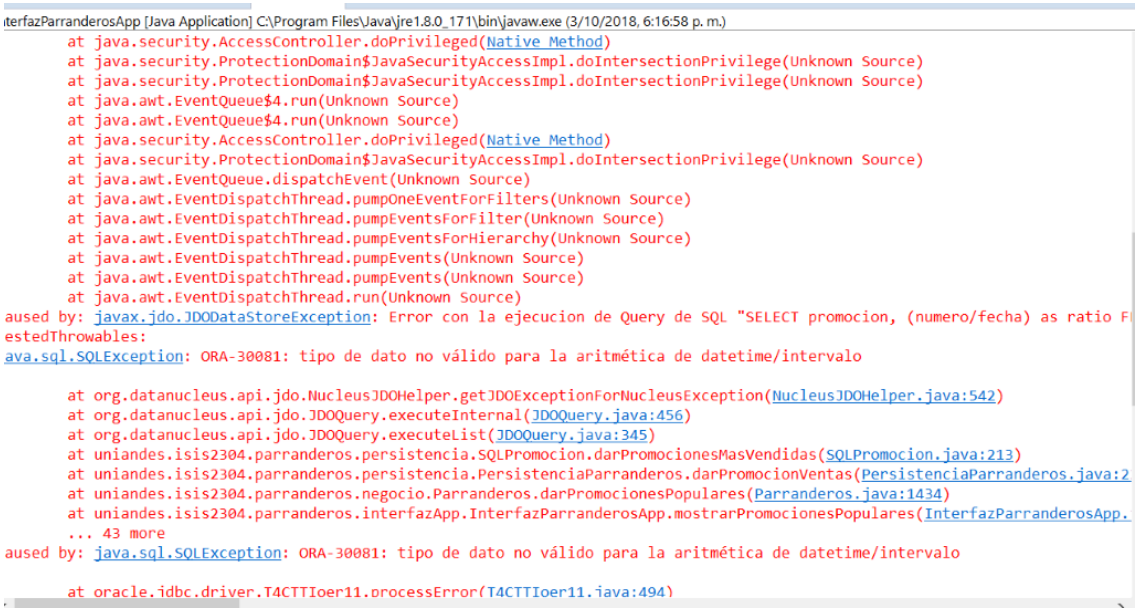


Figura 27. Captura de pantalla con error de requerimiento de consulta 2

En esta parte tuvimos un inconveniente con uno de los requerimientos de consulta en la interfaz, esto se debe a que el formato de la fecha no permitía que los datos del resultado del requerimiento se mostraran en consola por lo que el método no funcionaba. Sin embargo, en *SQL Developer* la sentencia funcionaba a la perfección mostrando resultados consistentes y acordes a la consulta.

Pruebas unitarias de las tablas:

Para la realización de las pruebas unitarias para las tablas, se realizaron 3 clases de *Tests*, el de conexión, el de supermercado y el de sucursal. Esto debido a que el hecho de realizar *tests* para las 14 clases (todas las tablas), no era práctico y por motivos de tiempo disponible tampoco era eficiente realizar esto.

Por esta razón, se probaron los mismos *tests* (unicidad e integridad) que estaban planeados para ser probados en todas las tablas, pero en las clases de Supermercado y de Sucursal con el fin de evidenciar los errores posibles en estos.

Para esta parte se tuvo el resultado de que los *test* de unicidad e integridad de Supermercado sirvieron y no tenían fallas. Sin embargo, el método `CRDSupermercadoTest` tenía fallas puesto a que los datos (escenario) que se creaba en los *tests*, se agregaban a los datos que ya estaban cargados en las bases de datos de *SQL Developer* y no se sobrescribían, por esta razón se tuvo que actualizar los valores esperado de acuerdo con el número de tuplas recalculado con los valores que ya estaban en las BD.

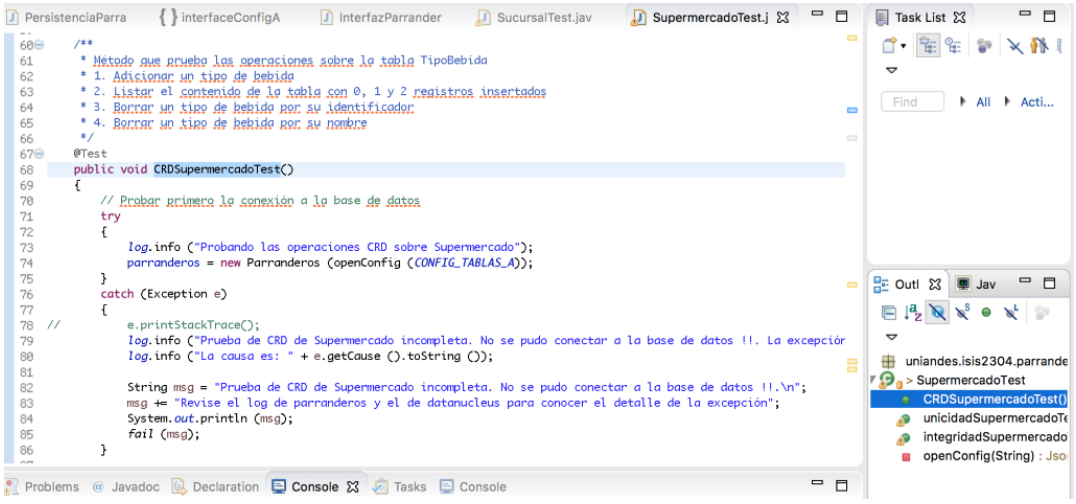


Figura 28. Captura de pantalla de clase SupermercadoTest

En la figura 28 se puede ver los métodos creados para el test de `Supermercado`, estos mismos fueron realizados para `Sucursal`.

El error fundamental que surgió a último minuto fue que el test se quedaba corriendo infinitamente y no sacaba un resultado. En la mañana los *tests* corrían (unicidad e integridad no CRD) y asociamos este error a los datos almacenados en la base de datos debido a que una de las excepciones que salían era de capacidad llena en memoria en la clase `LinkedList`.

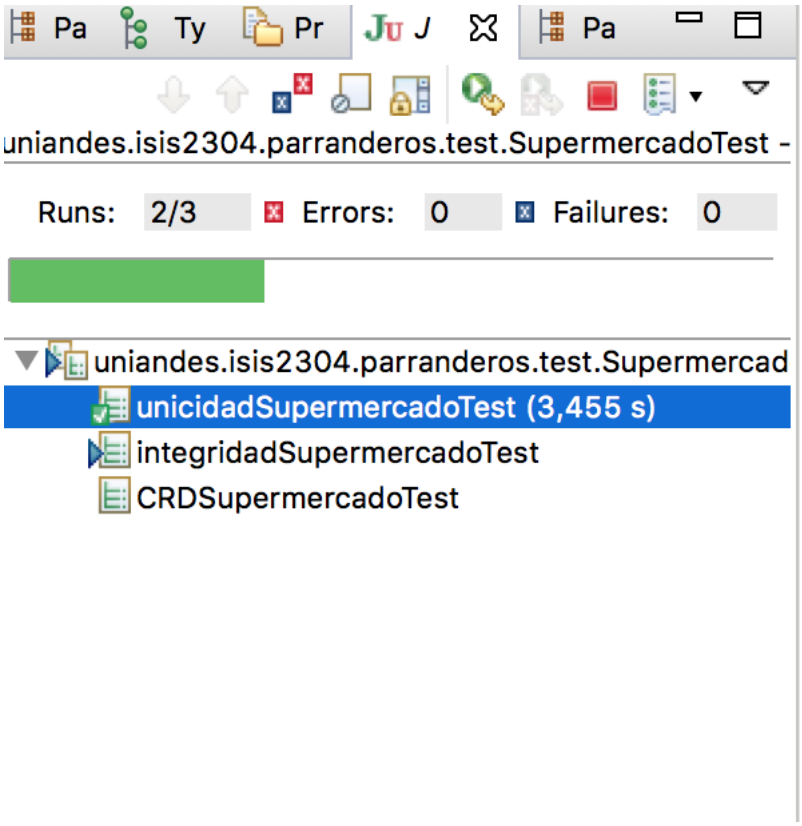


Figura 29. Captura de pantalla de resultados en curso de clase SupermercadoTest.

5 Balance plan de pruebas

Para la realización de las pruebas unitarias para las tablas, se realizaron 3 clases de *Tests*, el de conexión, el de supermercado y el de sucursal. Esto debido a que el hecho de realizar *tests* para las 14 clases (todas las tablas), no era práctico y por motivos de tiempo disponible tampoco era eficiente realizar esto.
Por esta razón, se probaron los mismos *tests* (unicidad e integridad) que estaban planeados para ser probados en todas las tablas, pero en las clases de Supermercado y de Sucursal con el fin de evidenciar los errores posibles en estos.

Haciendo un balance de las pruebas se puede establecer que tuvieron un funcionamiento del 66.6% debido a que de las 6 pruebas realizadas entre las clases `SupermercadoTest` y `SucursalTest`, 4 de estas funcionaron y 2 fallaron. Así mismo como grupo consideramos que hubiese sido mejor realizar más pruebas para las diferentes clases y comprobar estos errores. Sin embargo, como se comentaba anteriormente, el error en memoria vinculado a la base de datos, no hizo posible esto. Así mismo otra estrategia para mejorar los *tests* puede ser vaciar las tuplas existentes en las tablas y probar los *tests* con nuevos datos para no tener que actualizar los valores esperados en cada consulta de prueba realizada.

6 Análisis reglas de negocios

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

Una regla de negocio fundamental es la consistencia en cuanto a la pertenencia de los productos por parte de una sucursal. Se debe asegurar que todo producto que sea creado está asociado a una venta por parte de una sucursal y exclusivamente a una sola. Un producto en específico no puede ser vendido por más de una sucursal, esto traería problemas de consistencia en el mundo del negocio de SuperAndes.

Por otro lado, otra gran regla de negocio la abarcan los procesos transaccionales de la bodega, los estantes, los pedidos y la sucursal. Para el manejo de productos entre estas se debe verificar siempre la cantidad de existencias en bodega y estantes para poder realizar un pedido, se debe asegurar que nunca se sobrepase el nivel mínimo de productos para hacer el pedido y debe haber coherencia entre las bodegas y los estantes. Esto quiere decir que hay productos por cada estante, pero una bodega puede tener varios estantes con diferentes productos. Además, la bodega le pertenece exclusivamente a una sucursal y esta no puede estar compartida entre dos, ya que se violarían las reglas de negocio.

Es pertinente saber que para el desarrollo de nuestro proyecto, modelamos los subpedidos dentro de un pedido. Esto para facilitar la asociación que existe con el proveedor, debido a que una sucursal puede realizar varios pedidos a un proveedor. Esos pedidos no son únicamente de un producto, por lo que un pedido puede tener varios subpedidos con diferentes productos cada uno. Esto es sumamente importante que se mantenga debido a que no puede haber un subpedido con diferentes productos ya que se violarían las reglas de negocio y la implementación no funcionaría.

Por otro lado, encontramos que es de suma importancia mantener la coherencia en las transacciones de un cliente. Las facturas son únicas e intransferibles e identifican que un cliente efectivamente compró un producto, es por eso que mantener estos datos consistentes es muy importante ya que no se puede dar el caso de que una factura le pertenezca a dos

clientes, o que haya facturas duplicadas. Esta regla de negocio es sumamente importante para el funcionamiento de SuperAndes.

7 Conclusiones

- Desarrollamos habilidades en el proceso de diseño de una aplicación transaccional, a partir de la descripción de un caso de negocio , en el cual pudimos identificar los aspectos más importantes del mundo del negocio para así identificar las reglas de este.
- Desarrollamos habilidades en el proceso de generación de un modelo de datos relacional a partir de un modelo conceptual. Esto supuso ser un gran reto ya que el buen análisis del caso de estudio y planteamiento del modelo conceptual, determina si la aplicación en un futuro va a funcionar o no. El modelo relacional así mismo, permite la creación de las tablas en las bases de datos que en un futuro serán funcionales o no. Esto teniendo en cuenta las especificaciones y restricciones pertinentes ajustándose a las reglas de negocio.
- Incorporamos elementos de calidad del modelo de datos, con respecto a integridad de la información. 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. 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) 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 requerimientos funcionales y de consulta de una aplicación como lo fue SuperAndes. Todo esto teniendo en cuenta las reglas de negocio.
- Aprendimos a dividir el trabajo y convivir en grupos de desarrollo para poder sacar adelante un 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.

8 Bibliografía

- GARCIA-MOLINA, Hector, ULLMAN, Jeffrey, WIDOM, Jennifer.
"Database Systems: The complete Book". 2nd. Edition. Prentice Hall, 2009.
- CHURCHER, Claire. “Beggining Database Design. From novice to professional”. APress. 2007. Disponible en versión electrónica en la biblioteca Uniandes.