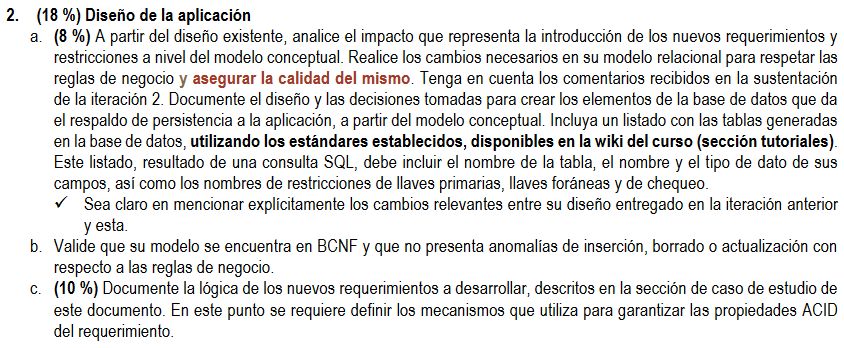
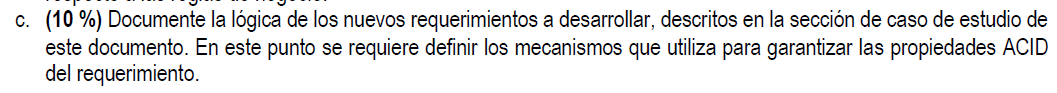


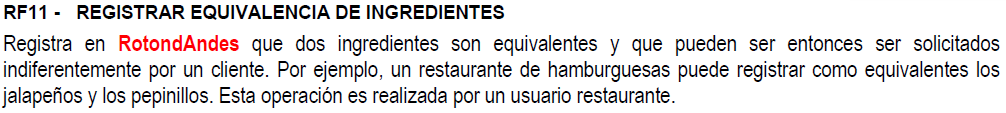
1. Para esta iteración hubo varios cambios en el modelo conceptual. Se comenzó con la creación de un atributo para modelar capacidades máximas de inventario a la hora de surtir los restaurantes. También, se cambió la forma de modelar las equivalencias entre ingredientes, que antes no estaba bien definida mediante el cambio de las cardinalidades del UML. Además, se creó relacionaron los ingredientes y los restaurantes, sin necesidad de que hubiera un producto de por medio (varios productos pueden pertenecer a varios restaurantes). Por último, para tener precisión en el registro de los pedidos se relacionaron los productos y los menús con los checkouts en una relación muchos a muchos.



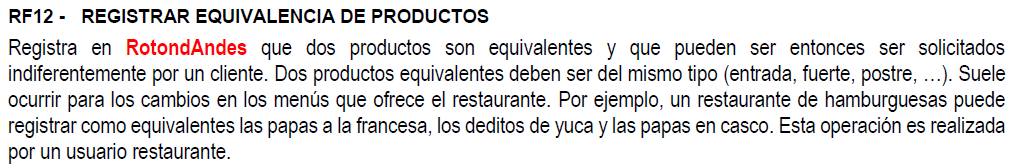
1. A. Como se vio, introducir los nuevos requerimientos no requiere de grandes cambios en el modelo conceptual sin embargo son vitales para garantizar la transaccionalidad en las operaciones que realice el sistema. Por ejemplo, antes para crear ingredientes no era posible hacerlo sin haber un producto entre el restaurante y el nuevo ingrediente, lo que no funciona si algún restaurante decide dejar un ingrediente sin un producto asignado. Del mismo modo, con el modelo anterior no era posible determinar equivalencias entre productos o ingredientes, cosa que se corrigió asignándole su correspondiente ‘categoría’ de equivalencia a cada ingrediente. También, a pesar de que el hecho de cambiar o crear las relaciones sea pequeño en el UML, en el modelo relacional cuesta más trabajo puesto que el tipo de la relación obliga a crear nuevas tablas, como es el caso de checkout con productos y menús.

B. Verificar la normalización de la tabla puede tornarse en una tarea costosa en términos de tiempo y documentación. Sin embargo decidimos analizar la tabla y concluimos que si está en la Forma Normal de Boyce-Codd (BCNF) debido a que para cada tabla (en la mayoría de casos), cada una de ellas solamente guarda la información con la que está relacionada con su llave principal (por lo que no es posible tener dependencias transitivas). Algunas tablas resultantes de la adaptación de ciertas relaciones en UML tienen llaves primarias compuestas, pero ninguna

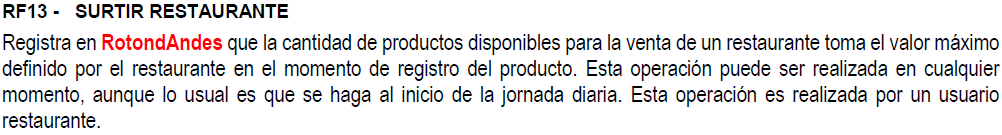




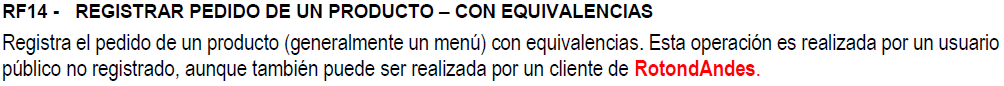
Este requerimiento se resuelve primero realizando la verificaciòn sobre el restaurante (login). Luego, es necesario determinar si los productos ya están registrados en una equivalencia. Si ambos están registrados se aborta notificando lo sucedido. En caso de que solamente haya uno, se registra el otro con el mismo número del que ya existe. Para el caso donde no existe ninguno, se tiene que el número de equivalencia siempre es mayor que cero y definimos que vamos a hacerlo creciente uno por uno. Por esta razón, buscamos en la tabla, para el respectivo restaurante , cual es el máximo de sus equivalencias, por último a la hora de registrar, el número de equivalencia para ambos es el máximo más uno.



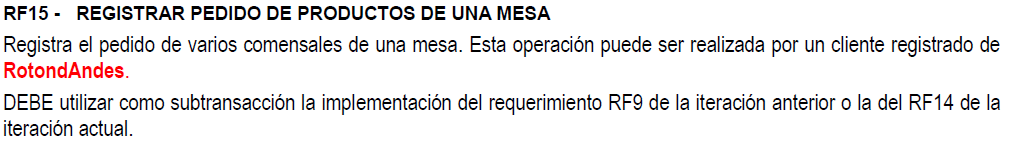
De manera muy similar al anterior, después de verificar la identidad del restaurante, se verifica el caso correspondiente para la información que exista en la tabla. El proceso es idéntico al del requerimiento anterior, solamente realizando el proceso con productos.



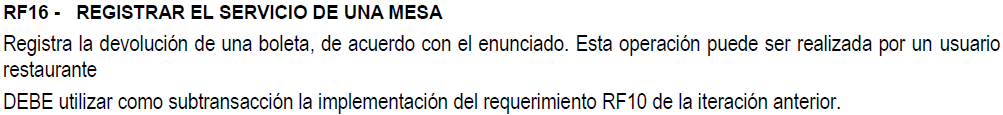
Primero, se verifica la identidad del restaurante. Después se realiza un update con autocommit en la tabla ProductosBodega volviendo la columna CantidadProducto el valor establecido en la columna Máximo, para el restaurante que lo solicitó.



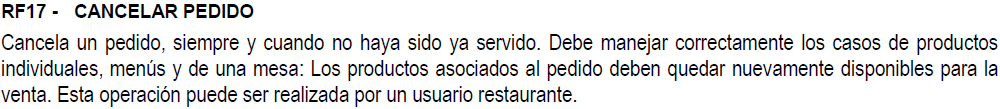
Como datos de entrada, definimos una orden como el conjunto de datos que contiene los datos de login del cliente, el nombre del menú o el producto, si es un menú o no; en caso de que sí, una lista con los productos alternativos del menú. Para este requerimiento, se realiza la verificación de los datos del cliente solamente si los datos de entrada relacionados al asunto no son nulos. Luego, en caso de que la orden contenga un menú, revisa la integridad de los productos escogidos, es decir, si son de diferentes categorías, porque un menú no tiene porductos de categorías iguales. Después, se busca si los escogidos son equivalentes con los productos originales del menú. Finalmente, si todo sale bien, se genera la orden de compra y el pedido ‘entra en espera’ esperando a que se descuente el producto de los inventarios de los restaurantes (esto se soluciona en el otro requerimiento).



Esta vez, se recibe un listado de ordenes como se definieron en el punto anterior. Se va a proceder de igual manera para cada orden, solamente que todas se van a guardar bajo el mismo checkout para así tener una manera de cancelar todos en caso de que haya que cancelar (deben cancelarse todos los del pedido original).



Como este requerimiento utiliza el RF10 de la itereación anterior, se explicará como funciona ésta. Cada pedido por mesa está guardado en un checkout, por lo que para servirlo basta con checkear que haya los suficientes productos de la orden en los restaurantes correspondientes, en caso de que todo salga bien, se marca como servido el checkout y se registra este tiempo. De lo contrario, se aborta la operación notificando en un mensaje de error y dejando el checkout de nuevo en estado de espera.



A la hora de cancelar un pedido, se elimina la orden correspondiente y todos los productos y menús que hacían parte de ella. Por supuesto antes de eso se verifica que el pedido no haya sido servido. Como antes de servirlo no se modifican ni se reservan los inventarios de los restaurantes, no es necesario corregir ningún dato relacionado.