Base de Datos II

Trabajo Práctico Obligatorio

Integrantes:

Bendayan, Alberto (Legajo: 62786)

Boullosa Gutierrez, Juan Cruz (Legajo: 63414)

Quian Blanco, Francisco (Legajo: 63006)

Stanfield, Theo (Legajo: 63403)

Profesores:

Rodriguez, Guillermo

Rodriguez, Cecilia

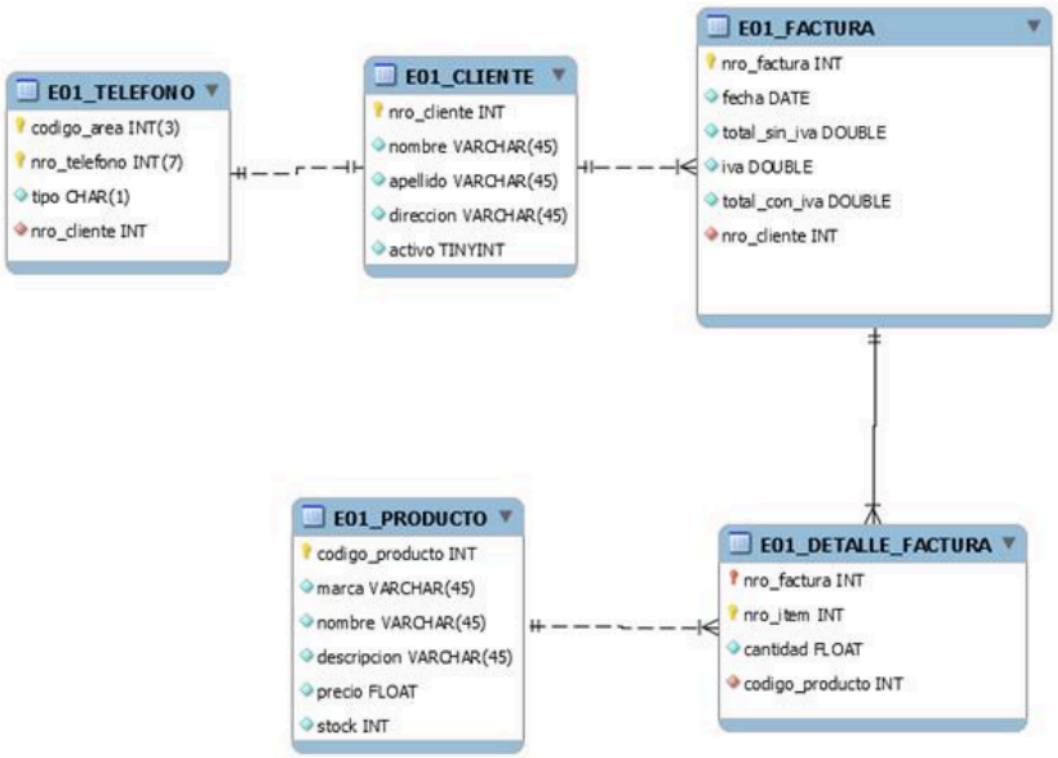
Decisión de las bases de datos

Las bases de datos NoSQL elegidas fueron MongoDB y Redis. Decidimos por MongoDB pues creemos que es una muy buena base de datos por guardar los esquemas en documentos JSON de los clientes y facturas. Nos permite embeber documentos dentro de otros lo cual nos ayuda muchísimo a la hora de tener relaciones 1;N y esto será aprovechado como se explicará más adelante a la hora de diseñar la estructura de los datos. Decidimos utilizar Redis como una base de datos de caching para acelerar la consulta a datos frecuentes. Decidimos que a la hora de tener que acceder a un mismo dato múltiples veces podría ser beneficioso reducir la cantidad de tiempo que conlleva realizar cada query.

Un factor importante a notar es que las consultas que piden una totalidad de los datos, como por ejemplo “todos los clientes que …” deberán buscar directamente en MongoDB ya que Redis, como es caché, los datos no siempre estarán disponibles. Los datos en Redis tienen un TTL (time to live) por lo que no asegura que todos los datos se encuentren en la base. Para asegurarnos que los datos devueltos sean todos los que cumplen las condiciones, se buscará directamente en MongoDB y las consultas que requieren de datos más específicos, se buscarán en Redis primero, y de no hallarlos, se buscarán en MongoDB.

Diseño y modelo de cada una de las bases

MongoDB se utilizará para guardar todos los datos y Redis se usará de cache para poder realizar algunas consultas con mayor rapidez. Partiendo del esquema DER provisto por la cátedra, realizamos las siguientes decisiones para modelar las bases:



Aprovechando la posibilidad de embeber documentos dentro de otros de MongoDB, decidimos embeber los teléfonos dentro de los clientes y de esta manera poder accederlos sin tener la necesidad de realizar un join.

De esta misma manera decidimos también, embeber los detalles de las facturas dentro de las misma facturas, así poder acceder a los productos utilizados de cada factura dentro de la misma.

Las colecciones final resultan de ser 3 (tres) y son las siguientes:

- clientes
- facturas
- productos

La implementación de Redis depende plenamente de los datos que tengamos que cachear, por lo que es importante leer las queries y observar atentamente qué datos son los que serán accedidos múltiples veces y tiene la necesidad de estar cacheados.

Las queries que requieren de datos específicos y no de un “todo”, son las queries 2 (dos) y 7 (siete) las cuales expresan lo siguiente

- Obtener el/los teléfono/s y el número de cliente del cliente con nombre “Jacob” y apellido “Cooper”
- Listar los datos de todas las facturas que hayan sido compradas por el cliente de nombre "Kai" y apellido "Bullock"

Por esta razones decidimos implementar dos estructuras de datos dentro de Redis para poder acelerar el comportamiento de estas dos queries. Las estructuras son las siguientes:

- Clave: **clientes:<nro_cliente>**, Valor: **<datos_cliente>**
- Clave: **clientes:names:<nombre><apellido>**, Valor: **<nro_cliente>**

La primera estructura sirve exclusivamente para poder acceder a los datos de los clientes dado su número de cliente, lo cual parece lógico realizar. Aquí surge un problema ya que como Redis no ofrece filtros de búsqueda, no podemos buscar por nombre. Nuestra solución a este problema fue crear la segunda estructura que separa por nombre y apellido de los clientes y nos permite encontrar su número de cliente. De esta manera, cuando en las queries 2 (dos) y 7 (siete) recibamos los nombres y apellidos de los clientes debemos buscar primero su número de cliente utilizando la segunda estructura mencionada. Una vez obtenido este número para la query 2 (dos) podremos buscar en la primera estructura mencionada anteriormente, los datos del cliente y para la query 7 (siete) ir directamente a mongoDB para buscar las facturas utilizando el número de cliente obtenido previamente.

La razón por la cual no implementamos el guardado de las facturas en Redis es debido a la gran magnitud que podrían llegar a tener estos datos. Consideramos que el guardado de datos en Redis debería ser exclusivamente para datos con tamaño controlado. Como Redis funciona en memoria no podemos guardar cualquier información dentro de él. Opinamos que las facturas son datos crecientes y pueden llegar a ser muy amplias en cuanto a su volumen. Tomando en cuenta todas las facturas de todos los clientes, la cantidad de datos se vuelve enorme, por lo que obviamos de implementar la estructura en Redis del tipo clave: **facturas:<nro_cliente>** y valor: **<datos_facturas>** y simplemente buscar estos datos directamente en mongoDB.

Respuestas a cada query

Obtener los datos de los clientes junto con sus teléfonos

Para esta consulta, dado que los teléfonos están embebidos dentro de la colección de clientes, fue suficiente con realizar una proyección en MongoDB que incluya los datos básicos del cliente y sus teléfonos. Como es una consulta que pregunta por un "todo", se consulta directamente en MongoDB, guardando en las colecciones de Redis los clientes en el proceso.

Obtener el/los teléfono/s y el número de cliente del cliente con nombre "Jacob" y apellido "Cooper"

Dado que tenemos las colecciones embebidas y los teléfonos están dentro del usuario, realizar esta consulta no fue complejo. Primero, nos fijamos si el cliente está en Redis, si no está lo buscamos en MongoDB y lo agregamos a Redis. Luego de eso, devolvemos el cliente en cuestión junto con sus teléfonos.

Mostrar cada teléfono junto con los datos del cliente

Esta consulta sigue una lógica similar a la anterior, pero en este caso, se desnormalizan los datos de manera que cada teléfono del cliente se representa como un registro individual. Esto se logra utilizando la operación de '*unwind'* en MongoDB para separar cada elemento de la lista de teléfonos en documentos distintos. Luego, se proyectan los datos del cliente junto con el teléfono correspondiente.

Obtener todos los clientes que tengan registrada al menos una factura

Para esta consulta, realizamos un '*lookup'* en mongo de nuestra colección de clientes con la colección de facturas. Poniendo como condición en el '*match'* que sólo devuelva aquellos que exista al menos una factura.

Identificar todos los clientes que no tengan registrada ninguna factura

Se realizó un procedimiento similar a la consulta del ítem anterior con la única diferencia que la consulta del '*match'* devuelve aquellos que no tengan ninguna factura registrada dado que se cambió la condición.

Devolver todos los clientes, con la cantidad de facturas que tienen registradas (si no tienen considerar cantidad en 0)

Se realizó un procedimiento similar a las consultas de los dos ítems anteriores, con el mismo '*lookup'*. A diferencia de las otras, esta query no usa el '*match'* ya que queremos todos los clientes y en la proyección de la misma agregamos otro campo '*facturas_count'* donde devolvemos la cantidad de facturas.

Listar los datos de todas las facturas que hayan sido compradas por el cliente de nombre "Kai" y apellido "Bullock"

Esta consulta consta de dos pasos. Primero intenta obtener el ID del cliente desde Redis. Si no lo encuentra, lo busca en MongoDB y guarda el resultado en Redis. Luego, usa el ID del cliente para buscar todas sus facturas en MongoDB y las imprime en formato JSON.

Seleccionar los productos que han sido facturados al menos 1 vez

En esta consulta, utilizamos un '*lookup*' entre la colección de facturas y la colección de productos. Posteriormente, aplicamos un '*unwind*' en la colección de facturas para desnormalizar los productos facturados. Esto descompone la lista de productos en documentos separados, permitiendo trabajar con cada producto de forma individual. Luego, se usa '*group*' para agrupar los productos, eliminando duplicados y asegurando que cada producto aparezca solo una vez. Finalmente, el '*match*' se usa para filtrar aquellos productos que hayan sido facturados al menos una vez, y el resultado es una lista de productos únicos que han aparecido en las facturas.

Listar los datos de todas las facturas que contengan productos de las marcas "Ipsum"

Esta consulta comienza con un '*lookup*' similar al de la consulta anterior, que une la colección de facturas con la de productos. Luego, se realiza un '*unwind*' para desnormalizar los productos incluidos en cada factura. A continuación, usamos un match para filtrar únicamente los productos cuya marca sea "Ipsum". Para garantizar que se devuelvan todas las facturas que contienen estos productos, no se realiza agrupación, ya que necesitamos mantener cada factura separada. Finalmente, se proyectan los datos de las facturas, mostrando la información relevante junto con los productos de la marca "Ipsum".

Mostrar nombre y apellido de cada cliente junto con lo que gastó en total, con IVA incluido

Primero, se realiza un lookup entre las colecciones de clientes y facturas. Luego, utilizamos '*unwind*' para desnormalizar las facturas y obtener una lista de todos los productos facturados a cada cliente. Después, en el '*group*', agrupamos por cliente (usando el ID del cliente), y sumamos el total gastado en las facturas, incluyendo el IVA. Si un cliente no tiene facturas, se devuelve un valor de 0. Finalmente, en la proyección, se incluyen los datos de nombre y apellido del cliente, junto con el total gastado (con IVA incluido).

Se necesita una vista que devuelva los datos de las facturas ordenadas por fecha

Se creó una vista en MongoDB que realiza un simple '*sort*' por el campo de fecha de las facturas. Esto asegura que los datos estén listos para ser consumidos directamente sin necesidad de procesamientos adicionales en la aplicación.

Se necesita una vista que devuelva todos los productos que aún no han sido facturados

Para esta consulta, se realizó un '*lookup'* entre las colecciones de productos y facturas, comprobando qué productos no aparecen en ninguna factura. Esto se logra utilizando un '*match'* con una condición que excluye aquellos productos que ya tienen referencias en facturas. El resultado final incluye únicamente los productos sin facturación.

Implementar la funcionalidad que permita crear nuevos clientes, eliminar y modificar los ya existentes

Para este ítem se realizaron 3 consultas:

- **Creación:** El creado que agrega el cliente a MongoDB y Redis. Para esto se realiza otra consulta a mongo para obtener el ultimo *nro_cliente* para poder saber cuál debemos utilizar par el nuevo cliente.
- **Modificación:** La modificación que actualiza un usuario a partir del *nro_cliente* esto actualiza ambas bases de datos, eliminando previamente el cliente desactualizado en Redis para mantener la consistencia.
- **Borrado**: Luego el borrado que a partir del *nro_cliente* borra el cliente de ambas bases de datos.

Implementar la funcionalidad que permita crear nuevos productos y modificar los ya existentes. Tener en cuenta que el precio de un producto es sin IVA

- **Creación:** Se genera un nuevo producto en MongoDB con los datos suministrados.
- **Modificación:** Se busca el producto en MongoDB utilizando su identificador único y se actualizan los campos necesarios (como el precio o descripción).