Universidad ORT Uruguay

Facultad de Ingeniería

Martin Alonso y Juan Carriquiry

Bases de datos 2

Segundo obligatorio doc parte 5

Grupo M5A
Docente: Cecilia Belleti
Docente: Tatiana Bellon

Formulario de Antecedentes
Curso Materia

Nro. Estudiante: **291799**
Nombres: **Martin**
Apellidos: **Alonso**
Grupo/Turno: **5A/Matutino**

Nro. Estudiante: **310190**
Nombres: **Juan**
Apellidos: **Carriquiry**
Grupo/Turno: **5A/Matutino**

1. Tabla de contenido

1) Pregunta 1

¿Cómo aplicaron los conceptos del material de estudio en su modelado de MongoDB?

En nuestro diseño de la colección personajes_progreso, aplicamos varios de los conceptos descritos en el artículo presente en la letra del obligatorio.

a) Conceptos aplicados:

i) Agregados:

El concepto más importante que utilizamos es el de Agregado. En lugar de normalizar los datos en múltiples tablas como haríamos en SQL, agrupamos toda la información relacionada con el progreso de un personaje en un único documento. Nuestro documento personajes_progreso es un Agregado que contiene no solo el ID del personaje, sino también un documento embebido con sus estadísticas y un array de documentos embebidos con sus logros. Esto es similar al ejemplo del "Producto" en el artículo.

ii) Denormalización:

Para lograr el patrón de Agregado, aplicamos la desnormalización. Por ejemplo, en lugar de tener una colección de "Logros" y referenciarlos por ID, copiamos la información de cada logro (nombre, descripción, tipo, xp) dentro del array logros de cada personaje que lo obtiene. Como menciona el artículo esto puede terminar significando que la misma información esté presente varias veces pero el beneficio es que aumenta la velocidad de lectura al hacer consultas.

iii) Agregados Atómicos:

Dado que toda la información de progreso de un personaje reside en un único documento, las actualizaciones a esa entidad son atómicas. Por ejemplo, cuando un personaje completa una misión y se le otorgan puntos de experiencia y un nuevo logro, ambas operaciones (actualizar el campo estadísticas. Misiones completadas y añadir un nuevo elemento al array logros) se pueden realizar en una sola operación atómica sobre ese único documento. Esto va de la mano con los anteriores conceptos.

iv) Jerarquías con Agregación de Árbol:

La forma en que modelamos la relación uno-a-muchos entre un personaje y sus logros puede considerarse una versión simple de la "Agregación de Árbol". Anidamos la colección de logros directamente dentro del documento

, lo que es muy eficiente para los casos de uso donde, como menciona el artículo, "el árbol se accede de una sola vez". En nuestro caso, al cargar el perfil de progreso de un jugador, siempre queremos ver todos sus logros.

Pero también se ve como en los casos del requerimiento 1 de la parte 4 que queremos filtrar por logros puede ser un poco menos eficiente ya que hay que hacer la etapa de Unwind y Group

2) Pregunta 2

¿Consideran que su modelado puede ser mejorado en algún aspecto?

Sí, consideramos que algunas de las posibles mejoras futuras son:

a) Mejora de Rendimiento con Índices:

Para las consultas de la Parte 4, especialmente la 4.1, que filtra logros por tipo, el rendimiento a gran escala podría degradarse. Una mejora sería crear un índice en el campo logros.tipo, lo cual mejoraría la vida útil de la BD a medida que el programa escala en usuarios.

b) Prevención del Crecimiento No Acotado:

Nuestro modelo asume que un personaje no tendrá un número infinito de logros. De cambiar esto (por ejemplo,en el caso de que los logros fueran eventos de log), el array logros podría crecer demasiado, llevando a documentos muy grandes y quitando rendimiento. La mejora que proponemos (mencionada en el artículo) sería cambiar a un modelo de Referencias (Application Side Joins). Para esto, crearíamos una segunda colección logros_obtenidos, donde cada documento representaría un único logro obtenido por un personaje, conteniendo una referencia al id del personaje.

3) Pregunta 3

a) Parte A

Considerando los conceptos vistos en el curso, comentar las ventajas y desventajas de haber utilizado MongoDB en este subsistema.

1) Ventajas:

a) *Esquema Dinámico y Flexibilidad:*
La principal ventaja fue cumplir el requisito de un sistema "altamente dinámico". Con el modelo diseñado,agregar un nuevo atributo a las estadísticas de los personajes es tan simple como ir al personaje que queremos modificar, entrar a su documento de estadísticas y modificarlo. Al estar en MongoDB no tenemos que preocuparnos por integridad de datos ni mantener las relaciones ni FK ni nada de eso

b) *Rendimiento en Lecturas*
(con excepciones mencionadas anteriormente): Gracias al patrón de Agregado, leer el perfil de progreso completo de un personaje es una única operación. Esto es significativamente más rápido que en un modelo relacional, donde necesitamos ejecutar JOINs entre tablas . El artículo enfatiza esta reducción de JOINs como un beneficio clave.

2) Desventajas:

a) *Transacciones Complejas:*
Mientras que nuestro modelo permite agregados atómicos, el modelo relacional de Oracle ofrece garantías ACID mucho más potentes y sencillas para transacciones que involucran múltiples entidades distintas. Como se menciona en el artículo, el soporte transaccional en NoSQL es a menudo más limitado.

b) *Consistencia y Duplicación de Datos:*
Por usar la desnormalización, si quisiéramos cambiar la descripción de un logro, tendríamos que buscar y actualizar cada documento de personaje que lo haya obtenido. En el modelo relacional, al estar normalizado, el cambio se haría en un único lugar (la tabla Logros), garantizando la consistencia.

b) Parte B

¿Encuentran algún otro subsistema o caso de uso que podría haberse usado este tipo de base de datos?
Sí, en el campo de los videojuegos modernos es compatible en varios casos con estos tipos de sistemas.

1) *Tablas de Liderazgo en Tiempo Real:*
   Para un ranking de los jugadores con más enemigos derrotados que se actualice constantemente, una base de datos clave-valor en memoria como Redis con su estructura de "Sorted Sets" sería órdenes de magnitud más rápida que cualquier consulta a una base de datos en disco.

2) *Sistema Social de Gremios y Amistades:*
   Las complejas interconexiones entre jugadores (quién es amigo de quién, quién pertenece a qué team, quién es enemigo de quién) forman un grafo. Una base de datos de grafos como Neo4j sería la herramienta perfecta para este subsistema, permitiendo consultas como "encontrar amigos de mis amigos que jueguen en mi misma zona horaria" de una forma que sería extremadamente compleja en SQL.