DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

21 de Noviembre de 2015 — Bases de Datos

Integrante	LU	Correo electrónico
García, Diego	223/97	diego.garcia.mail@gmail.com
Morales, Marcelino	14/02	marcelino.morales@gmail.com
Schmit, Matías	714/11	matias.schmit@gmail.com
Tastzian, Juan Manuel	39/10	jm@tast.com.ar

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Introducción

En el siguiente informe

2. Desnormalización

Para la resolución de este ejercicio entregamos el diseño físico de la DB realizado en formato JSON con el agregado de que en lugar de indicar un valor para cada atributo indicamos el tipo de datos. De esta forma hacemos mas legible el modelo. Además discutimos cada uno de los puntos de optimización pedidos comentando las distintas variantes que evaluamos para la resolución de los mismos. Para la definición de los tipos de datos a utilizar, al no estar especificados en el DER, los elegimos de la forma que nos pareció mas relevante al contexto.

El diseño de la DB propuesto es el siguiente:

```
db: {
Clientes: [{
DNI : int,
Nombre : string,
Edad : int,
CantCompras : int,
CantComprasMismaEdad : int
}],
Empleados: [{
NroLegajo,
Nombre,
ClientesAtendidos:[{
DNI : int,
Edad : int,
Nombre : string,
Fecha : string
}],
Sectores: [{
CodSector: string,
IdTarea: int
}]
}],
Articulos: [{
CodBarras : string,
Nombre : string,
CantVendidos: int,
CodSector : string
}],
Sectores: [{
CodSector: string,
Trabaja: [{
NroLegajo,
IdTarea
}]
}],
Tareas: [{
IdTarea: int,
Descripcion: string
}]
}
```

2.1. Resolución de Consultas

2.1.1. Empleados que atendieron a clientes de mayor edad

Para resolver esta consulta embebimos información de los clientes dentro del documento de Empleados, agregando una lista de clientes atendidos. De esta forma, al obtener el documento del empleado correspondiente podemos acceder directamente a los datos de todos los clientes que fueron atendidos

por él, y por lo tanto también podemos hacer un filtro sobre la entidad Empleados que devuelva los empleados que atendieron a los clientes mayores de edad.

2.1.2. Los articulos mas vendidos

Para resolver esta consulta se nos ocurren 3 opciones:

1. Embeber la información de compra dentro del artículo. El problema con este enfoque es que se replicaría la información del cliente en cada compra, y además la consulta pedida queda muy compleja de hacer.
2. Mantener una lista de artículos comprados dentro de cada cliente. Esto reduce la replicación de información, pero sigue sin solucionar la complejidad de la consulta pedida.
3. Finalmente, optamos por reutilizar la idea del punto 2 y agregar un contador de ventas dentro de cada artículo. Esto nos permite resolver la consulta de manera eficiente, con mínima redundancia, sin perder la información del modelo.

2.1.3. Los sectores donde trabajan exactamente 3 empleados

La consulta se resuelve buscando qué sectores tienen exactamente 3 elementos en el array "Trabaja". Esto vale porque la aridad de la ternaria no permite que haya un empleado de un sector con más de 1 tarea asignada

Para resolver esta consulta embebimos una parte de la información perteneciente a la entidad Empleado dentro del documento Sector, de esta forma resolvemos eficiéntemente la consulta (que se realiza contando la cantidad de elementos del array Trabajapor cada sector) y mantenemos replicada solo la información necesaria para resolverla.

2.1.4. El empleado que trabaja en más sectores

Para esta consulta decicimos guardar en cada empleado una lista de los sectores en los que trabaja. De esta forma podemos resolver esta consulta contando la longitud del array de sectores de cada empleado, y quedándonos con el de mayor longitud.

2.1.5. Ranking de los clientes con mayor cantidad de compras

Para optimizar al máximo esta consulta optamos por mantener un contador de compras dentro del documento de clientes. De esta forma resulta trivial el armado de un ranking de clientes con mayor cantidad de compras. Además esta opción solo requiere que al realizar una venta actualicemos el contador de compras del cliente.

2.1.6. Cantidad de compras realizadas por clientes de la misma edad

Para resolver esta consulta decidimos agregar un contador de compras extra a la entidad cliente que cuenta el número de compras realizadas por clientes de esa edad. De esta forma por cada compra realizada, debemos incrementar el contador de todos los clientes que tengan la misma edad que el contador. Ésto nos permite mantener un nivel óptimo de redundancia y evitar soluciones mas desprolijas como crear una tabla de edades por ejemplo.

3. Map Reduce

4. Sharding

4.1. Funcion de creación de datos

Se utilizo el siguiente código en js, para generar las inserciones de datos, permitiendo definir un total y un tamaño de lote (y una pausa si fuera necesario). La función genera registros para insertar,en este caso únicamente con código postal aleatorio (ya que era lo importante para el ejemplo) e imprime la información provista por los comandos que provee mongodb para reportar el estado del sharding **sh.status()** y **db.collection.getShardDistribution()**.

```
function insertRandomPersonas(total, tam_intervalo, pausa_seg) {

  var start = new Date().getTime();
  var total_count = total;

  var cp_size = 99999 ;
  print ("Inicial");
  sh.status()
  db.personas.getShardDistribution()
  print ("--------------------------------------------------------------------------------");

  while (total_count>0){
    var current_loop = Math.min(total_count, tam_intervalo);
    for (var i = 1; i <= current_loop; i++) {

      var randomCP = Math.floor((Math.random() * cp_size) );

      db.personas.insert({"nombre" : "Juan Perez" ,"password" : "1234" ,"codigo_postal"
          : randomCP  ,"genero" : "m" , "edad" : 28 , "fecha_creacion": "2015-01-01"} )
    }
    total_count = total_count - current_loop;
    var end = new Date().getTime();
    var time = end - start;
    print ("Restantes:"+total_count+" Execution time: " + time);
    sh.status()
    db.personas.getShardDistribution()
    print ("--------------------------------------------------------------------------------");
    sleep(pausa_seg * 1000);
  }
}
```

La llamada para insertar 5000000 registros en rangos de 20000 es la siguiente:

```
insertRandomPersonas(500000,20000,0);
```

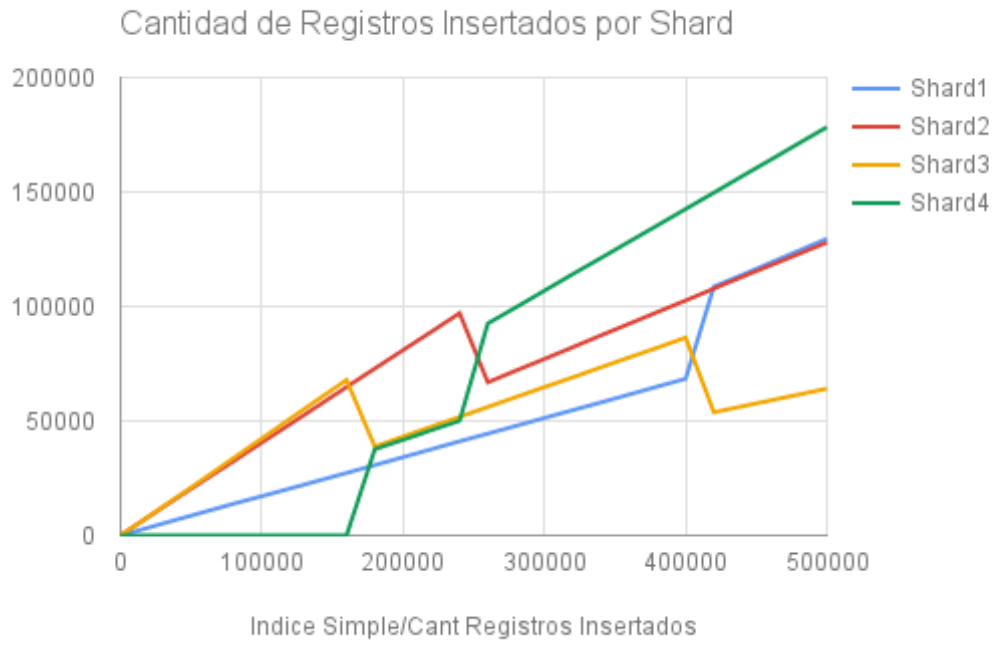
4.2. Pruebas

Para las pruebas a realizar se configuraron 4 shards de acuerdo al tutorial provisto por la cátedra y se realizaron pruebas primero utilizando un *indice simple* y después (regenerando de cero los shards) un *indice hash*.
Se realizaron varias pruebas para cada tipo de indice, sin embargo se provee solo un ejemplo de cada una, ya que los resultados por tipo de indice tuvieron el mismo comportamiento, sin tener exactamente las mismas cantidades insertadas por el componente aleatorio en la insercion.

4.3. Resultados

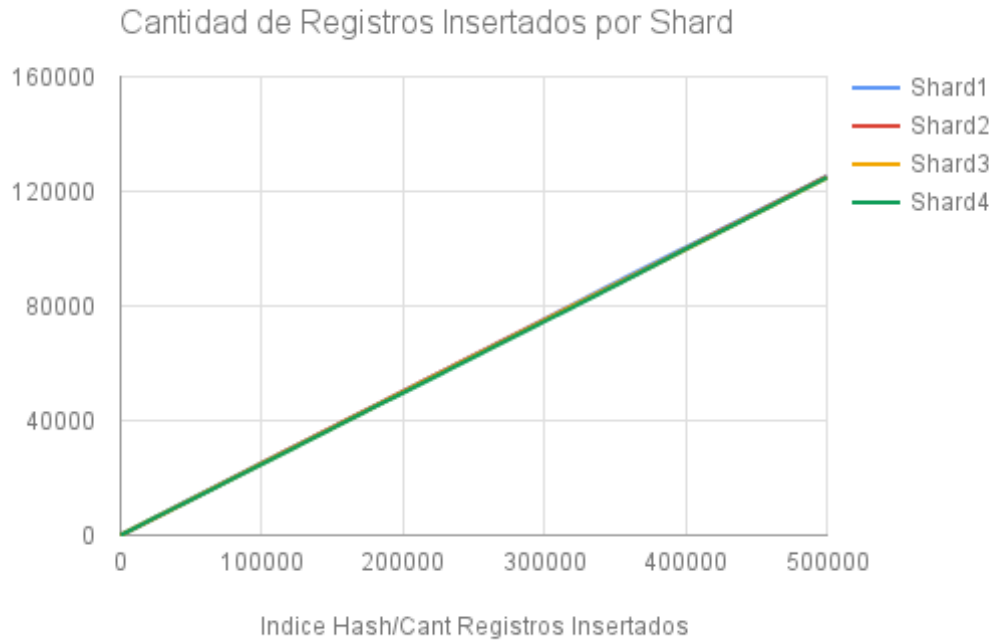
Indice Simple

Cant Registros Insertados	Shard1	Shard2	Shard3	Shard4
0	0	0	0	0
20000	3441	8237	8322	0
40000	6925	16346	16729	0
60000	10362	24416	25222	0
80000	13829	32392	33779	0
100000	17128	40546	42326	0
120000	20478	48579	50943	0
140000	23888	56671	59441	0
160000	27289	64815	67896	0
180000	30705	72895	38764	37636
200000	34168	80927	43157	41748
220000	37626	88957	47486	45931
240000	41100	97070	51754	50076
260000	44453	66860	56108	92579
280000	47847	72029	60453	99671
300000	51300	77048	64775	106877
320000	54697	82171	69025	114107
340000	58072	87357	73420	121151
360000	61526	92442	77729	128303
380000	65013	97555	82002	135430
400000	68423	102615	86393	142569
420000	108673	107816	53816	149695
440000	113798	112928	56349	156925
460000	119068	117956	58953	164023
480000	124353	122915	61521	171211
500000	129591	127964	64092	178353



Indice Hash

Cant Registros Insertados	Shard1	Shard2	Shard3	Shard4
0	0	0	0	0
20000	5039	4901	5067	4993
40000	10170	9942	9967	9921
60000	15211	14962	15003	14824
80000	20109	20089	19971	19831
100000	25155	25113	24940	24792
120000	30143	30194	29923	29740
140000	35116	35160	34950	34774
160000	40093	40220	39926	39761
180000	45084	45172	44947	44797
200000	50160	50210	49953	49677
220000	55212	55214	54972	54602
240000	60194	60226	59949	59631
260000	65222	65226	64931	64621
280000	70210	70247	69921	69622
300000	75309	75162	74942	74587
320000	80403	80062	80020	79515
340000	85527	84939	84987	84547
360000	90583	89902	89914	89601
380000	95566	94898	94837	94699
400000	100428	99931	99805	99836
420000	105363	105137	104749	104751
440000	110394	110037	109801	109768
460000	115318	115106	114785	114791
480000	120196	120176	119788	119840
500000	125288	125160	124708	124844



4.4. Analisis

Se puede ver que utilizando un *indice simple* para un campo en el cual los datos estan generados de manera aleatoria, lo que sucede es que empieza a crecer un shard mas que otros y en algun momento el balanceador decide migrar un subconjunto de estos datos a otro shard menos cargado. Esto se va realizando varias veces durante la carga de datos, de acuerdo al nivel de crecimiento de cada shard.
Una vez realizadas las pruebas verificamos cual es el mecanismo utilizado por mongodb en estos casos y esta explicado en la seccion **Chunk Migration Across Shards** del manual de mongodb. Básicamente decide balancear cuando la diferencia entre la cantidad de *chunks* entre el shard con mas *chunks* y el shard con menos *chunks* es superior a un valor configurado internamente dentro del balanceador de mongodb.
Utilizando un *indice hash* lo que sucede es que los registros generados se van insertando de manera pareja entre todos los shards y en ese caso nunca le hace falta al balanceador hacer una migración de *chunks*.

4.5. Escenarios posibles para sharding

Algunos escenarios posibles para sharding son:

Tener una base con consultas/visitas de clientes a un sitio web con alta concurrencia, que guarden que items vio, en que esta interesado, historial de navegación, etc. Estas consultas/visitas pueden crecer muy rápidamente y generar un volumen my grande de datos. En este caso una base con mongodb con sharding permitiría un crecimiento acorde al uso que va teniendo la aplicación.Un atributo posible para sharding en este caso es el numero de cliente.

Tener una empresa de logística o correo, que quiere mantener una base con el total de envíos de paquetes realizados, en este caso se podría utilizar como atributo de sharding código postal del destinatario, siempre y cuando el envío de paquetes realizados tenga una distribución uniforme (o lo mas próxima posible a uniforme) y variada de destinos para los envíos con los que trabaja.En este caso la base podría crecer lo que fuera necesario y sabríamos que tendríamos equilibrada entre todos los shards.

Queremos tener una base de datos de usuarios conectados a un juego online de alta concurrencia. Esta base debe guardar información variada de los jugadores y de las partidas mientras están conectados. Esta base puede tener variaciones en la carga dependiendo de los días del horario, día de la semana o época del año (en vacaciones tiene mas uso). En este caso se podría usar el sharding para permitir un crecimiento/decrecimiento de la base de acuerdo a lo que fuera necesario y de manera equilibrada. En este caso podríamos usar como atributo para sharding el id de jugador.

4.6. Características de un atributo para sharding

Un atributo para ser usado como clave en un esquema de sharding debe tener algunas características:

- Alta Cardinalidad: Si un atributo tiene baja cardinalidad no es útil para sharding, ya que todos los registros con el mismo atributo terminan en un mismo chunk y en caso de que uno quisiera tener un numero grande de shards, esta acotado por la cardinalidad de este atributo.
- Tener una distribución uniforme: Un atributo debe tener una distribucion uniforme para el dominio del problema en el que se esta trabajando, ya que si no es posible que se llenen mucho algunos shards y otros queden vacios. Por ejemplo utilizar código postal como clave para shard puede ser bueno (inclusive con los balanceos) si tiene una distribución pareja (tenemos personas o clientes de todo el país), sin embargo si nuestra base de clientes es de un solo barrio de una ciudad, la variabilidad de códigos postales puede ser muy acotada.

5. Otras bases de datos NoSQL

6. Conclusiones