Departamento de Computación,
Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires

Trabajo Práctico 2

Bases de Datos

Primer Cuatrimestre de 2016

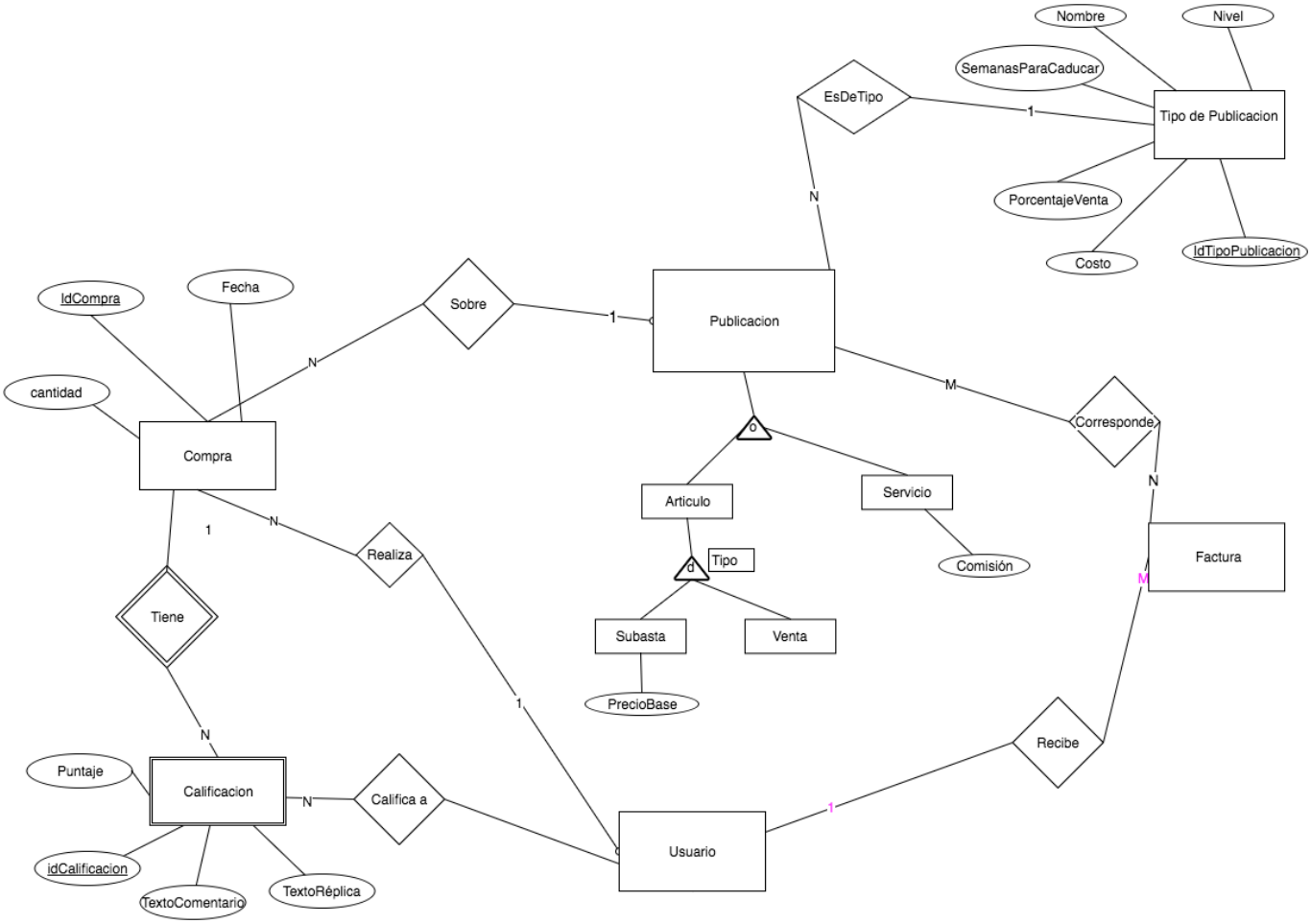
Apellido y Nombre	LU	E-mail
Federico Hosen	825/12	federico.hosen@gmail.com
Martin 'Marto' Caravario	470/12	martin.caravario@gmail.com
Guido Rajngewerc	379/12	guido.raj@gmail.com
Christian Russo	679/10	christian.russo8@gmail.com

1. Introducción

En el siguiente trabajo se desarrolla un modelo de Bases de Datos NoSQL con MongoDB teniendo en cuenta el problema descripto en el TP1. Para esto, se utilizó una sección del DER completo del TP1 y se diseñaron los documentos correspondientes para realizar las consultas pedidas en el enunciado del TP2.

2. Parte 1 - Diseño

2.1. Diagrama Entidad relacion



Las consultas pedidas solamente necesitan de usuarios, ventas, reputaciones, calificaciones, comisiones y facturas. Por lo tanto usamos:

- Para usuarios y ventas, las tablas de Usuarios, Publicaciones y Compra
- Para reputaciones de usuarios, las tablas de Calificación y Usuario
- Para comisiones, usamos las tablas de Publicación y Tipo de Publicación.
- Para monto de facturas, usamos las tabla de Facturas

2.2. Colecciones

A continuación, mostramos las colecciones que utilizamos para resolver las consultas propuestas en el enunciado

2.2.1. Publicaciones

```
{
  "Id": 65,
  "TipoPublicacion": "Servicio"
}
```

2.2.2. Compra

```
{
  "idCompra": 656897,
  "idUsuarioComprador": 65468,
  "Fecha": "2-06-2014",
  "Cantidad": 2,
  "Publicacion": {
    "idPublicacion": 2365,
    "idUsuario": 123,
    "Precio": 3655,
    "PorcentajeVenta": 20 //entre 0 y 100
  },
  "CalificacionDelVendedor": 8,
  "CalificacionDelComprador": 6
}
```

2.2.3. Factura

```
{
  "idFactura": 687468,
  "IdUsuario": 654,
  "Fecha": "16-9-2015",
  "TotalAPagar": 5624,
  "estoyPagando": [
    {
      "idPublicacion": 54,
      "TipoSuscripcion": "Rubi"
    },
    {
      "idPublicacion": 54,
      "TipoSuscripcion": "Libre"
     }
  ]
}
```

2.3. Migración datos SQL a Colecciones

Para migrar los datos de SQL (MySQLWorkwench) al formato de JSON que se necesita en MongoDB lo que hicimos fue exportar los datos directo desde MySQLWorkwench, pero estos se exportan en JSON plano, es decir no quedaba formateado de la forma que nosotros queriamos.

Para solucionar este problema, realizamos unos algoritmos en python para formatearlos a mano. Estos algoritmos reciben como input el JSON plano y devuelven un JSON formateado para MongoDB

2.3.1. Parser para el JSON de la compra

```
import json
from pprint import pprint

with open(raw_input()) as data_file:
    data = json.load(data_file)
for a in data:
  print "{"
  print '"idCompra": ' , a['idCompra'], ","
  print '"idUsuario": ', a['idUsuario'],","
  print '"fecha": ', '"',a['fecha'],'"',","
  print '"cantidad": ', a['cantidad'],","
  print '"Publicacion": {'
  print '"idPublicacion": ', a['idPublicacion'],","
  print '"idUsuario": ', a['idUsuario'],","
  print '"Precio": ', a['precio'],","
  print '"PorcentajeVenta": ', a['porcentajeVenta']
  print "}" ","
  print '"calificacionVendedor": ', a['CalificacionDelVendedor'], ','
  print '"calificacionComprador": ', a['CalificacionDelComprador'], ''
  print '}'
```

2.3.2. Parser para el JSON de la Factutra

```
import json
from pprint import pprint

with open('factura_sql.json') as data_file:
    data = json.load(data_file)

c = {}
for a in data:
  c.setdefault(a["idFactura"],[]).append(a)

for a in c:
  print '{'
  print '"idFactura": ', c[a][0]['idFactura'], ","
  print '"IdUsuario": ',c[a][0]['idUsuario'], ","
  print '"Fecha": ', '"',c[a][0]['fecha'],'"', ","
  total = 0
  for x in c[a]:
    total = total + c[a][c[a].index(x)]['totalAPagar']
  print '"TotalAPagar": ',total, ","
  print '"estoyPagando": ['
  i = 1
  for x in c[a]:
    print "{"
    print '"idPublicacion":', c[a][c[a].index(x)]['idPublicacion'], ","
    print '"TipoSuscripcion":','"',c[a][c[a].index(x)]['nombre'], '"'
    if i == len(c[a]):
      print "}"
    else:
      print "},"
    i = i+1
  print ']'
  print '}'
```

Nota: para la colección de publicaciones no tuvimos que hacer ningún algoritmo.

2.4. Consultas SQL

A continuación listamos las consultas realizadas para conseguir los datos necesarios para nuestras colecciones:

2.4.1. SQL de Publicaciones

```
select p.idPublicacion,
CASE
  when s.idPublicacion is not null and
  (v.idPublicacion is not null or su.idPublicacion is not null) then "mixto"
  when s.idPublicacion is not null then "servicio"
    when v.idPublicacion is not null then "articulo"
    when su.idPublicacion is not null then "articulo"
END
as TipoPublicacion
from publicacion p
left join servicio s on s.idPublicacion = p.idPublicacion
left join venta v on v.idPublicacion = p.idPublicacion
left join subasta su on su.idPublicacion = p.idPublicacion
```

2.4.2. SQL de Compras

```
select c.idCompra, c.idUsuario as idUsarioComprador,p.idUsuario as idUsuarioVendedor,
c.fecha, c.cantidad, c.idPublicacion, p.precio,
tp.costo, tp.porcentajeVenta, tp.nombre, ca.puntaje
from compra c
join publicacion p

on c.idPublicacion = p.idPublicacion
join tipo_de_publicacion tp
on p.idTipoDePublicacion = tp.idTipoPublicacion
join calificacion ca
on c.idCompra = ca.idCompra
```

2.4.3. SQL de Facturas

```
select f.idFactura, p.idUsuario, f.fecha, f.totalAPagar, p.idPublicacion, tp.nombre

from factura f

inner join corresponde c
on c.idFactura = f.idFactura

inner join publicacion p
on c.idPublicacion = p.idPublicacion

inner join tipo_de_publicacion tp
on tp.idTipoPublicacion = p.idTipoDePublicacion
```

2.5. Implementación MongoDB

Para implementar la base de datos de documentos en mongoDB utilizamos los JSON de las colecciones descriptos anteriormente y los importamos de la siguiente forma:

```
mongoimport --db tp2 --collection compras --drop --file ./compra_mongo.json
mongoimport --db tp2 --collection facturas --drop --file ./factura_mongo.json
mongoimport --db tp2 --collection publicacion --drop --file ./publicacion_mongo.json
```

3. Parte 2 - Map Reduce

3.1. Ejercicio 1

El importe total de ventas por usuario.

```
var ej1_m = function(){
  emit(this.Publicacion.idUsuario, this.Publicacion.Precio * this.cantidad)
}

var ej1_r = function(k, vs){
  return Array.sum(vs);
}
```

3.2. Ejercicio 2

La reputación histórica de cada usuario según la calificación.

```
var ej2_m = function(){
  if(this.calificacionVendedor != null) emit(this.idUsuario, this.calificacionVendedor);
  if(this.calificacionComprador != null) emit(this.Publicacion.idUsuario, this.calificacionComprador);
}

var ej2_r = function(k, vs){
  return ( Array.sum(vs) / vs.length);
}
```

3.3. Ejercicio 3

Las operaciones con comisión más alta.

```
var ej3_m = function(){
  var getComision = function(compra){
   var subs = compra.Publicacion;
    return (subs.Precio * compra.cantidad) * (subs.PorcentajeVenta / 100);
  };

  emit("todos", {
  'idCompra': this.idCompra,
  'comision': getComision(this)});
};

var ej3_r = function(k, vs){
  var max = Math.max(...vs.map(function(v){
      return v.comision;
  }));
  return {"resultado": vs.filter(function(v){
    return v.comision >= max;
  }), "max": max};
};
```

3.4. Ejercicio 4

El monto total facturado por año.

```
var ej4_m = function(){
  var getAno = function(fecha){
    return fecha.trim().substring(0,4);
  }
  emit(getAno(this.Fecha), this.TotalAPagar)
}

var ej4_r = function(k, vs){
  return Array.sum(vs);
}
```

3.5. Ejercicio 5

El monto total facturado por año de las operaciones pertenecientes a usuarios con suscripciones Rubi Oriente.

```
var ej5_m1 = function(){

  var esRuby = function(e, i, arr){
    return e.TipoSuscripcion.trim() == "RubiDeOriente"
  };

  var getAno = function(fecha){
    return fecha.trim().substring(0,4);
  }

  emit(
    {
      "ano": getAno(this.Fecha),
      "usuario": this.IdUsuario
    },
    {
      "totalAPagar": this.TotalAPagar,
      "esRuby": this.estoyPagando.some(esRuby)
    });
};

var ej5_r1 = function(k, vs){
  if (vs.some(function(a){
    return a.esRuby;
  }))
  {
    return Array.sum(vs.map(function(e){return e.totalAPagar}));
  }
}


var ej5_m2 = function(){
  if (this.value.esRuby)
  {
    emit(this._id.ano, this.value.totalAPagar);
  }
}

var ej5_r2 = function(k, vs){
  return Array.sum(vs);
}
```

3.6. Ejercicio 6

El total de publicaciones por tipo de publicación (productos, servicios o mixtas).

```
var ej6_m = function(){
  emit(this.TipoPublicacion, 1);
}

var ej6_r = function(k,vs){
  return  vs.length;
};
```

Listing 1: Ejercicio 6

4. Parte 3 - Sharding

4.1. Introducción

Utilizamos 5 shards para hacer las pruebas y a la colección de publicaciones le agregamos un atributo extra ïdUsuarioVendedor"para poder usarlo como atributo clave a la hora de hacer experimentos de sharding. Cada 200000 insert hacemos un output en un archivo de

db.publicaciones.getShardDistribution() y sh.status() y con estos datos hicimos los gráficos.

4.2. Inserción de datos en la base de datos

Para insertar 500000 registros en la base de datos lo que hicimos fue generar un script en Python que nos devuelva 500000 publicaciones.

El pseudocodigo para la generación de los 500000 registros:

```
import json
from pprint import pprint
import random
from random import randint


foo = ['articulo', 'servicio', 'mixto']
print 'var pubs = ['
for x in range(1,500000):
  print '{'
  print '"idPublicacion":', x,","
  print '"TipoPublicacion":', ",",random.choice(foo),","
  print '"idUsuarioVendedor":',randint(0,8000)
  print '},'
print ']'
```

A continuación, para insertar estos datos en la base de datos de mongo, hicimos un scrpit de JavaScript para correrlo desde el Shell de mongo y que pueda importar todos estos datos en los distintos shards

```
var collection = "publicaciones";

pubs.forEach(function(e, i){
    if (i % 20000 == 0){
        db[collection].getShardDistribution();
    }
    db[collection].insert(e);
});
```

Listing 2: Ejercicio 6

5. Índice Simple

5.1. Creación de Shards

Para la creacion de los Shards con Indice Simple corrimos los siguientes scripts:

Creamos los shards:

```
mkdir -p ./data/localhost10000
mongod --rest --shardsvr --port 10000 --dbpath data/localhost10000
--logpath data/localhost10000/log

mkdir -p ./data/localhost10001
mongod --rest --shardsvr --port 10001 --dbpath data/localhost10001
--logpath data/localhost10001/log

mkdir -p ./data/localhost10002
mongod --rest --shardsvr --port 10002 --dbpath data/localhost10002
--logpath data/localhost10002/log

mkdir -p ./data/localhost10003
mongod --rest --shardsvr --port 10003 --dbpath data/localhost10003
--logpath data/localhost10003/log

mkdir -p ./data/localhost10004
mongod --rest --shardsvr --port 10004 --dbpath data/localhost10004
--logpath data/localhost10004/log
```

Creamos el config server

```
mkdir -p ./data/localhost10005
mongod --rest --configsvr --port 10005 --dbpath data/localhost10005
--logpath data/localhost10005/log
```

Creaamos routing service

```
 mongos --port 10006 --configdb localhost:10005 > run_routing_service_log
```

Agregamos los shards

```
mongo localhost:10006
use admin
db.runCommand({addshard:"localhost:10000", name:"shard10000"});
db.runCommand({addshard:"localhost:10001", name:"shard10001"});
db.runCommand({addshard:"localhost:10002", name:"shard10002"});
db.runCommand({addshard:"localhost:10003", name:"shard10003"});
db.runCommand({addshard:"localhost:10004", name:"shard10004"});
```

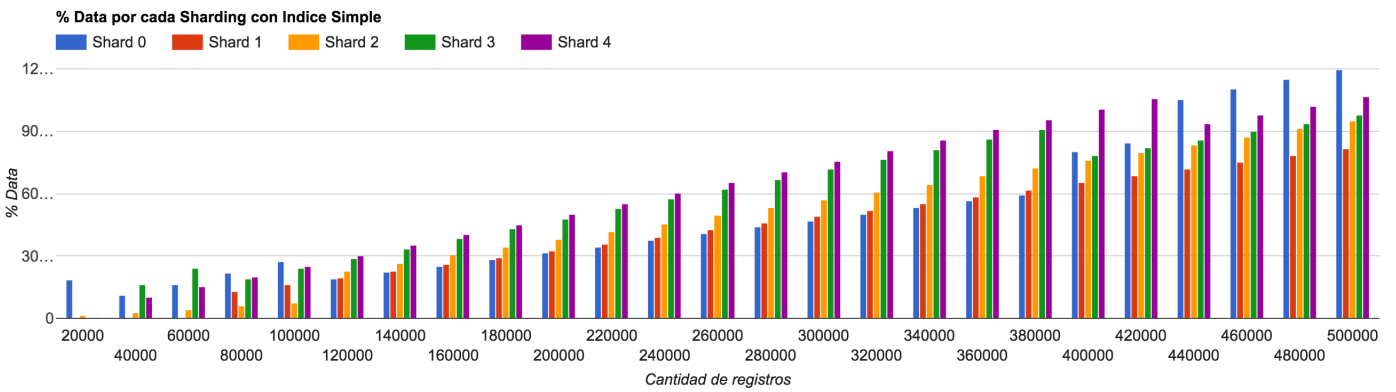
Configuramos los Shards:

```
use test_sharding
sh.enableSharding("test_sharding")
db.publicacion.ensureIndex({"idUsuarioVendedor": 1})
```

5.2. Evolucion de Shards

A continuación se muestra un cuadro donde se encuentran los datos obtenidos del análisis de los Shards. Lo que hicimos fue ver el getShardDistribution() cada 20000 inserciones y mostrar el porcentaje de distrubución de los datos en los distintos Shards. Como ya fue explicado, utilizamos 5 Shards y en el cuadro se ve el porcentaje de datos que contiene cada Shard cada 20000 insert desde 0 hasta 500000 registros. Para este análisis utilizamos índices simples sobre la columna ïdUsuarioVendedor"del documento de Publicaciones.

	Shard 0	Shard 1	Shard 2	Shard 3	Shard 4
20000	**18566**	12	1422	0	0
40000	**10898**	27	2895	16012	10168
60000	**16399**	33	4389	24017	15162
80000	**21962**	13045	5879	19014	20100
100000	**27395**	16295	7256	23916	25138
120000	**18870**	19565	22801	28630	30134
140000	**22053**	22790	26591	33464	35102
160000	**25124**	26010	30487	38234	40145
180000	**28224**	29276	34306	43027	45167
200000	**31345**	32547	38098	47838	50172
220000	**34452**	35822	41816	52637	55273
240000	**37547**	39089	45576	57523	60265
260000	**40700**	42449	49379	62116	65356
280000	**43853**	45730	53114	66861	70442
300000	**46953**	48882	56887	71705	75573
320000	**50053**	52090	60751	76535	80571
340000	**53176**	55353	64618	81312	85541
360000	**56417**	58495	68396	86047	90645
380000	**59516**	61838	72166	90935	95545
400000	**80161**	65164	75885	78202	100588
420000	**84135**	68400	79757	82031	105677
440000	**105287**	71633	83544	85904	93632
460000	**110087**	74902	87305	89798	97908
480000	**114876**	78190	91144	93638	102152
500000	**119624**	81454	94919	97576	106427



6. Indice Hash

6.1. Creacion de Shards

Para la creacion de los Shards con Indice Hash corrimos los siguientes scripts:

Creamos los shards:

```
mkdir -p ./data/localhost10000
mongod --rest --shardsvr --port 10000 --dbpath data/localhost10000
--logpath data/localhost10000/log

mkdir -p ./data/localhost10001
mongod --rest --shardsvr --port 10001 --dbpath data/localhost10001
--logpath data/localhost10001/log

mkdir -p ./data/localhost10002
mongod --rest --shardsvr --port 10002 --dbpath data/localhost10002
--logpath data/localhost10002/log

mkdir -p ./data/localhost10003
mongod --rest --shardsvr --port 10003 --dbpath data/localhost10003
--logpath data/localhost10003/log

mkdir -p ./data/localhost10004
mongod --rest --shardsvr --port 10004 --dbpath data/localhost10004
--logpath data/localhost10004/log
```

Creamos el config server

```
mkdir -p ./data/localhost10005
mongod --rest --configsvr --port 10005 --dbpath data/localhost10005
--logpath data/localhost10005/log
```

Creaamos routing service

```
 mongos --port 10006 --configdb localhost:10005 > run_routing_service_log
```

Agregamos los shards

```
mongo localhost:10006
use admin
db.runCommand({addshard:"localhost:10000", name:"shard10000"});
db.runCommand({addshard:"localhost:10001", name:"shard10001"});
db.runCommand({addshard:"localhost:10002", name:"shard10002"});
db.runCommand({addshard:"localhost:10003", name:"shard10003"});
db.runCommand({addshard:"localhost:10004", name:"shard10004"});
```

Configuramos los Shards:

```
use test_sharding
sh.enableSharding("test_sharding")
db.publicacion.ensureIndex({"idUsuarioVendedor": "hashed"})
```

6.2. Evolución de Shards

A continuación se muestra un cuadro donde se encuentran los datos obtenidos del análisis de los Shards. Lo que hicimos, fue ver el getShardDistribution() cada 20000 inserciones y mostrar el porcentaje de distrubución de los datos en los distintos Shards. Como ya fue explicado, utilizamos 5 Shards y en el cuadro se ve el porcentaje de datos que contiene cada Shard cada 20000 insert desde 0 hasta 500000 registros. Para este análisis utilizamos índices hash sobre la columna ïdUsuarioVendedor"del documento de Publicaciones.

	Shard 0	Shard 1	Shard 2	Shard 3	Shard 4
20000	3970	3957	3934	4134	4005
40000	7979	7933	8018	7995	8075
60000	12000	11891	12002	12022	12085
80000	15999	15799	15993	16092	16117
100000	19968	19877	20008	20013	20134
120000	24017	23810	24072	24035	24066
140000	28113	27737	27992	28042	28116
160000	32055	31792	31936	32022	32195
180000	35999	35911	35940	35964	36186
200000	39889	39965	40000	39951	40195
220000	43965	44059	43872	43942	44162
240000	47961	48022	47859	48034	48124
260000	51904	52054	51932	52046	52064
280000	55854	56078	55858	56038	56172
300000	59820	60113	59859	60026	60182
320000	63791	64051	63867	64192	64099
340000	67683	68115	67831	68248	68123
360000	71707	72019	71812	72294	72168
380000	75701	76031	75853	76242	76173
400000	79715	80052	79889	80249	80095
420000	83766	83884	83915	84373	84062
440000	87698	87908	87984	88335	88075
460000	91684	91946	92108	92223	92039
480000	95652	96077	96045	96236	95990

7. Indice Simple vs Indice Hash

En el caso de shards utilizando indice hash se puede ver que la distribución es casi ideal. A medida que se van agregando documento, van creciendo la cantidad de documentos en cada shard pero siempre de forma uniforme.

Mientras que en el caso de shards utilizando indices simples se puede ver que luego de la primera corrida casi el 100 % de la información esta en el shard numero 1, puede que ésto se deba a que la cantidad de entradas en es muy baja. Luego se empiezan a utilizar los otros shard pero con una distribución no tan regular, creciendo cada shard de forma proporcional hasta que, a partir de las 400.000 entradas el shard 1 empieza a aumentar en mayor medida su contenido, manteniendoce así hasta el final.

En base a esta experimientación, parecería que es mas seguro confiar en los índices de tipo hash, ya que distribuyen de forma casi ideal el contenido de la colección sobre los distintos shards.

Sin embargo, el set de datos no es tan rico como para realizar muchas pruebas, queda pendiente para un futuro explorar ésta técnica de particionado con más datos y probar con índices compuesto, y con un conjunto de consultas establecidas, para analizar la performance de éstas.

8. Posibles escenarios de Sharding

- Un posible escenario seria uno en el cual tenemos una base de datos muy grande y con una distribucion uniforme de datos de todo el mundo. Para eso se podria definir un atributo continente, lo cual nos permitiria dividir la base en diversos shards (uno en cada continente) para poder trabajar con una menor cantidad de datos. Esto traeria la ventaja de que ante la caida de un nodo de un continente no se perderian los datos de todo el mundo y solo habria que recuperar ese continente determinado. Esta escenario no seria recomendado en caso de tener muchos datos sobre un continente determinado, ya que estariamos sobrecargando ese nodo en particular.
- Otro posible escenario seria uno en el cual tengamos una base de datos muy grande sobre personas con distribucion uniforme sobre sus apellidos,(es decir sobre las letras con las que comienza cada apellido). En este caso podriamos crear una letra como atributo y asi particionar en shards nuestra base, ubicando en cada servidor las personas cuyos apellidos comienzan con determinada letra. Esto trae la ventaja de que trabajermos en cada servidor con una menor cantidad de personas, en particular con las personas cuyo apellido comienza con la letra del shard correspondiente. Esto nuevamente es posible en el escenario en el cual tengamos una distribucion uniforme sobre la primer letra de cada apellido, pues de lo contrario estariamos sobrecargando ese nodo.
- Un tercer escenario posible en cual usar un atributo para sharding podria ser uno en el cual tengamos una base de datos sobre personas en un hospital y decidamos definir un atributo que sea el factor sanguineo. Esto permitira dividir nuestra base de datos en shards, en el cual cada nodo tendra un factor sanguineo en particular. Esto agilizara la consulta a la hora de resolver una emergencia medica, ya que a la hora de resolver una consulta solo debera buscarse en ese nodo particular y no en toda la base, por ejemplo si se quisieran buscar donantes de sangre para un factor en particular.

9. Caracteristicas atributos en Sharding

El objetivo es elegir una clave que nos permita que:

- Todos los inserts, updates y deletes deberían estar distribuidos uniformemente sobre los shards.
- Todas las consultas deberían ser distribuidas de forma uniforme sobre los shards.
- Todas las operaciones deberían ejecutarse sólo sobre un shard, es decir, no se debería hacer un delete sobre un shard que no contiene al dato.
- Una consulta no debería hacerse sobre un shard que no tiene los datos que están siendo buscados.

Teniendo en cuenta ésto, un atributo usado para sharding debería:

- (Cardinality) Poder ser subdividido en rangos, para poder tener varios chunks dentro de cada shard
- (Write Distribution) Como describimos más arriba, se desea distribuir las escrituras de forma uniforme. Para ésto una clave computada con algun nivel de aleatoriedad (MD5 or SHA1).

- (Read Distribution) Para que las consultas estén distribuidas de forma uniforme, antes que nada hay que saber qué tipo de consultas la bd va a manejar, con lo cual no se puede saber de forma general la forma de la clave para asegurar ésto. Pero una opción que podría funcionar en muchos casos es tener una clave compuesta por algún campo de alta granularidad (por ejemplo un id, o un hash) junto con algún atributo que sea usado en la consulta, por ejemplo, si busco datos por un año, el año podría formar parte de la clave.
- (Read Targeting) Como dijimos anteriormente, nos gustaría que una consulta se ejecute sólo sobre un shard. Para ésto es neesario que la clave de sharding esté presente en la consulta, con lo cual, uno debería elegir una clave que esté presente en la mayoría de las consultas que se necesiten responder rápido.
- (Read Locality) Este ítem aplica a las consultas por rangos, como en Read Targeting, nos gustaría que la consulta se ejecute sólo sobre un shard. Para ésto lo que se suele hacer es crear una clave compuesta que tenga, por ejemplo, un id y una fecha, si se busca por rango de fechas.