



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA



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 'Martó' Caravario	470/12	martin.caravario@gmail.com
Guido Rajngewerc	379/12	guido.raj@gmail.com
Christian Russo	679/10	christian.russo8@gmail.com

2.2. Colecciones

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

2.2.1. Publicaciones

```
1 {
2   "Id": 65,
3   "TipoPublicacion": "Servicio"
4 }
```

2.2.2. Compra

```
1 {
2   "idCompra": 656897,
3   "idUsuarioComprador": 65468,
4   "Fecha": "2-06-2014",
5   "Cantidad": 2,
6   "Publicacion": {
7     "idPublicacion": 2365,
8     "idUsuario": 123,
9     "Precio": 3655,
10    "PorcentajeVenta": 20 //entre 0 y 100
11  },
12   "CalificacionDelVendedor": 8,
13   "CalificacionDelComprador": 6
14 }
```

2.2.3. Factura

```
1 {
2   "idFactura": 687468,
3   "IdUsuario": 654,
4   "Fecha": "16-9-2015",
5   "TotalAPagar": 5624,
6   "estoyPagando": [
7     {
8       "idPublicacion": 54,
9       "TipoSuscripcion": "Rubi"
10    },
11    {
12       "idPublicacion": 54,
13       "TipoSuscripcion": "Libre"
14    }
15  ]
16 }
```

2.3. Migracion 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 Facutra

```
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 coleccion de publicaciones no tuvimos que hacer ningun algoritmo.

2.4. Consultas SQL

A continuacion 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)
  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 idUsuarioComprador, p.idUsuario as idUsuarioVendedor, c.fecha, c.cantidad
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. Implementacion 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.

```
1 var ej1_m = function(){
2   emit(this.Publicacion.idUsuario, this.Publicacion.Precio * this.cantidad)
3 }
4
5 var ej1_r = function(k, vs){
6   return Array.sum(vs);
7 }
```

3.2. Ejercicio 2

La reputación histórica de cada usuario según la calificación.

```
1 var ej2_m = function(){
2   if(this.calificacionVendedor != null) emit(this.idUsuario, this.calificacionVendedor);
3   if(this.calificacionComprador != null) emit(this.Publicacion.idUsuario, this.calificacionComprador);
4 }
5
6 var ej2_r = function(k, vs){
7   return ( Array.sum(vs) / vs.length);
8 }
```

3.3. Ejercicio 3

Las operaciones con comisión más alta.

```
1 var ej3_m = function(){
2   var getComision = function(compra){
3     var subs = compra.Publicacion;
4     return (subs.Precio * compra.cantidad) * (subs.PorcentajeVenta / 100);
5   };
6
7   emit("todos", {
8     'idCompra': this.idCompra,
9     'comision': getComision(this)});
10 };
11
12 var ej3_r = function(k, vs){
13   var max = Math.max(...vs.map(function(v){
14     return v.comision;
15   }));
16   return {"resultado": vs.filter(function(v){
17     return v.comision >= max;
18   }), "max": max};
19 };
```

3.4. Ejercicio 4

El monto total facturado por año.

```
1 var ej4_m = function(){
2   var getAno = function(fecha){
3     return fecha.trim().substring(0,4);
4   }
5   emit(getAno(this.Fecha), this.TotalAPagar)
6 }
7
8 var ej4_r = function(k, vs){
9   return Array.sum(vs);
10 }
```

3.5. Ejercicio 5

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

```
1 var ej5_m1 = function(){
2
3   var esRuby = function(e, i, arr){
4     return e.TipoSuscripcion.trim() == "RubiDeOriente"
5   };
6
7   var getAno = function(fecha){
8     return fecha.trim().substring(0,4);
9   }
10
11   emit(
12     {
13       "ano": getAno(this.Fecha),
14       "usuario": this.IdUsuario
15     },
16     {
17       "totalAPagar": this.TotalAPagar,
18       "esRuby": this.estoyPagando.some(esRuby)
19     });
20 };
21
22 var ej5_r1 = function(k, vs){
23   if (vs.some(function(a){
24     return a.esRuby;
25   })))
26   {
27     return Array.sum(vs.map(function(e){return e.totalAPagar}));
28   }
29 }
30
31
32 var ej5_m2 = function(){
33   if (this.value.esRuby)
34   {
35     emit(this._id.ano, this.value.totalAPagar);
36   }
37 }
38
39 var ej5_r2 = function(k, vs){
40   return Array.sum(vs);
41 }
```

3.6. Ejercicio 6

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


```
1 var ej6_m = function(){
2   emit(this.TipoPublicacion, 1);
3 }
4
5 var ej6_r = function(k,vs){
6   return vs.length;
7 };
```

Listing 1: Ejercicio 6

4. Parte 3 - Sharding

4.1. Introduccion

Utilizamos 5 shards para hacer las pruebas y a la coleccion de publicaciones le agregamos un atributo extra `idUsuarioVendedor` 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 graficos.

4.2. Inserccion 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 generacion 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 continuacion, para insertar estos datos en la base de datos de mongo, hicimos un script de JavaScript para correrlo desde el Shell de mongo y que pueda importar todos estos datos en los distintos shards

5. Indice Simple

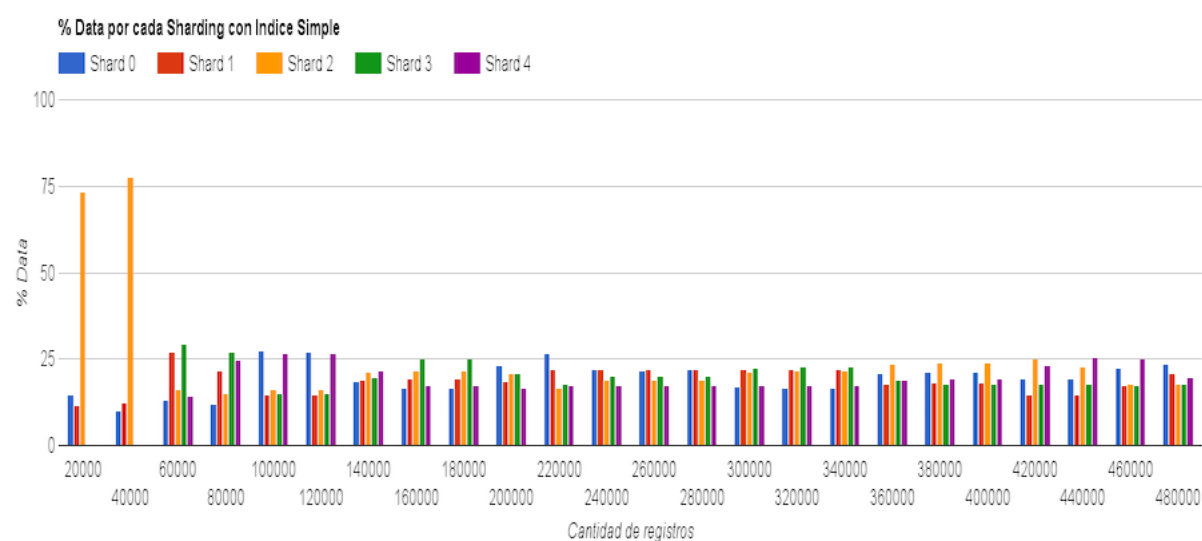
5.1. Creacion de Shards

Poner como hicimos para crear los shards

5.2. Evolucion de Shards

A continuacion se muestra un cuadro donde se encuentran los datos obtenidos del analisis de los Shards. Lo que hicimos fue ver el getShardDistribution() cada 20000 inserciones y mostrar el porcentaje de distribucion 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 analisis utilizamos indices simples sobre la columna idUsuarioVendedor"del documento de Publicaciones.

	Shard 0	Shard 1	Shard 2	Shard 3	Shard 4
20000	14,86	11,66	73,47	0	0
40000	10,2	12,32	77,47	0	0
60000	13,06	27,03	16,22	29,27	14,39
80000	11,95	21,61	14,91	26,88	24,64
100000	27,26	14,59	16,24	15,05	26,83
120000	27,17	14,61	16,32	15,12	26,76
140000	18,4	18,91	21,16	19,88	21,63
160000	16,65	19,31	21,61	25,05	17,35
180000	16,62	19,31	21,66	25,01	17,37
200000	23,06	18,63	20,9	20,69	16,68
220000	26,48	21,88	16,59	17,68	17,34
240000	21,83	21,92	18,91	20,04	17,28
260000	21,81	21,92	18,89	20,05	17,3
280000	21,82	21,91	18,88	20,06	17,3
300000	16,95	21,84	21,41	22,52	17,26
320000	16,6	21,93	21,51	22,62	17,32
340000	16,6	21,94	21,49	22,63	17,31
360000	20,85	17,71	23,5	18,87	19,04
380000	21,15	17,96	23,84	17,7	19,32
400000	21,15	17,96	23,82	17,69	19,35
420000	19,19	14,78	25,07	17,68	23,26
440000	19,21	14,69	22,84	17,65	25,58
460000	22,33	17,52	17,72	17,3	25,1
480000	23,7	20,92	17,72	17,83	19,8



En el grafico se puede ver que

6. Indice Hash

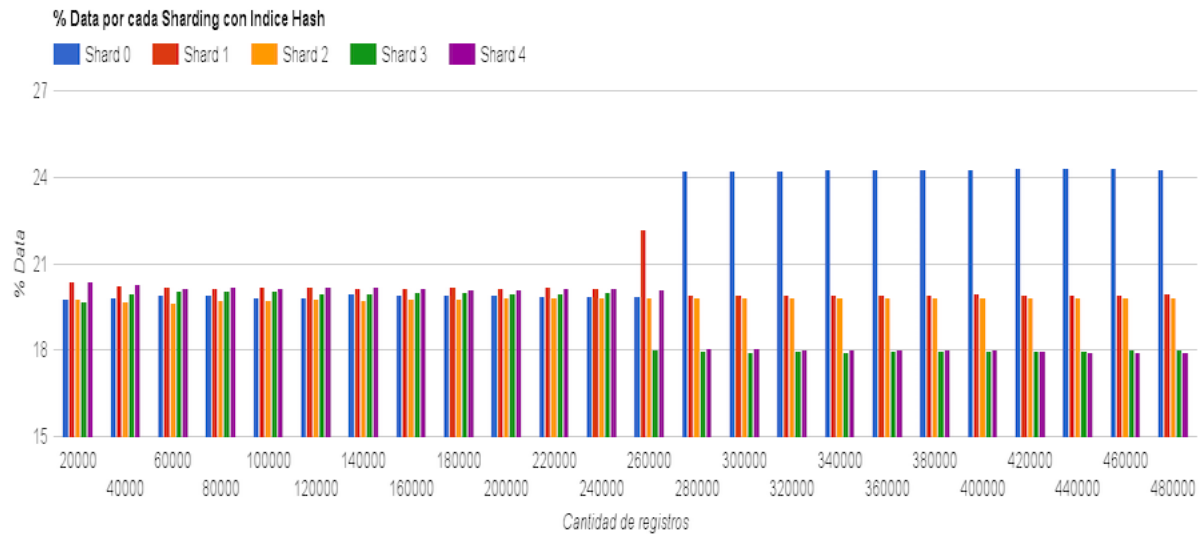
6.1. Creacion de Shards

Poner como hicimos para crear los shards

6.2. Evolucion de Shards

A continuacion se muestra un cuadro donde se encuentran los datos obtenidos del analisis de los Shards. Lo que hicimos fue ver el `getShardDistribution()` cada 20000 inserciones y mostrar el porcentaje de distribucion 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 analisis utilizamos indices hash sobre la columna `idUsuarioVendedor` del documento de Publicaciones.

	Shard 0	Shard 1	Shard 2	Shard 3	Shard 4
20000	19,77	20,38	19,77	19,68	20,38
40000	19,82	20,22	19,68	19,98	20,27
60000	19,93	20,19	19,64	20,06	20,15
80000	19,9	20,12	19,71	20,05	20,2
100000	19,84	20,21	19,74	20,05	20,14
120000	19,84	20,18	19,78	19,98	20,2
140000	19,94	20,16	19,74	19,96	20,17
160000	19,93	20,12	19,77	19,99	20,16
180000	19,92	20,19	19,76	19,99	20,11
200000	19,92	20,16	19,84	19,95	20,11
220000	19,87	20,17	19,83	19,98	20,12
240000	19,87	20,15	19,83	20,01	20,12
260000	19,86	22,16	19,83	18,02	20,1
280000	24,21	19,93	19,84	17,95	18,04
300000	24,22	19,93	19,83	17,94	18,06
320000	24,23	19,92	19,83	17,96	18,03
340000	24,26	19,93	19,82	17,94	18,03
360000	24,27	19,93	19,82	17,97	17,99
380000	24,27	19,92	19,82	17,97	17,99
400000	24,28	19,94	19,82	17,95	17,99
420000	24,31	19,92	19,81	17,97	17,96
440000	24,31	19,93	19,82	17,97	17,94
460000	24,3	19,92	19,82	18	17,94
480000	24,28	19,94	19,81	18	17,94



En el grafico se puede ver que

7. Indice Simple vs Indice Hash

aca escribir la conclusion