DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico II

10 de noviembre de 2015

Bases de datos

Segundo Cuatrimestre de 2015

Integrante	LU	Correo electrónico
Ignacio Truffat	837/10	el_truffa@hotmail.com
Gaston Rocca	836/97	gastonrocca@gmail.com
Agustín Godnic	689/10	agustingodnic@gmail.com
Matías Pizzagalli	257/12	matipizza@gmail.com

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. Desnormalización.

Pequeña intro al problema...

1.1. Empleados que atendieron clientes mayores de edad.

embebimos los clientes dentro de los empleados, con la fecha de atención, quedando nos asi:

```
Empleado: {
  nroLegajo: int,
  nombre: string,
  clientes: [{DNI, Nombre, Edad, Fecha}],
  ...
}
```

Luego, para responder la consulta deseada hay que correr:

```
db.empleados.aggregate(
  [
    { $unwind: "$clientes" },
    { $match: {"clientes.Edad": { $gt: 17 } }  },
    { $group: { _id: "$nombre", nombre:{$first:"$nombre" } } },
    { $project : { _id:0, nombre: 1 } }
  ]
)
```

1.1.1. Ejemplo

Para insertar registros en la base corremos:

```
db.empleados.insert( { nroLegajo: 003, nombre: "Ernestino Juanes",
  clientes: [ {DNI: 40528343,  Nombre: "Raul Juan Lopez", Edad: 14,
  Fecha: "14/03/2015"} ] } )
db.empleados.insert( { nroLegajo: 002, nombre: "Juan Paez",
  clientes: [ {DNI: 30154820, Nombre: "Juana Perez", Edad: 23,
  Fecha: "20/04/2015"}, {DNI: 40528753, Nombre: "Raul Lopez", Edad: 15,
  Fecha: "23/04/2015"} ] } )
db.empleados.insert( { nroLegajo: 001, nombre: "Joaquina Paez",
  clientes: [ {DNI: 30154820,  Nombre: "Juan Perez", Edad: 25,
  Fecha: "03/04/2015"} ] } )
```

Luego, una vez insertados los registros deseados, corremos la consulta mencionada arriba y nos da:

```
{ "nombre" : "Joaquina Paez" }
{ "nombre" : "Juan Paez" }
```

1.2. Articulos mas vendidos.

embebimos los DNIs de clientes q compraron dentro de los articulos y buscamos los maximos.

```
Articulos: {
  CobBarras: int,
  nombre: string,
  compradores: [{DNI}]
}
```

Asumimos que hay un unico articulo mas vendido:

```
db.articulos.aggregate(
  [
    { $unwind : "$Compradores"},
    {$group : { _id: "$CodBarras", CodBarras:{$first:"$CodBarras"},
      Nombre:{$first:"$Nombre"}, totalVendidos: {$sum: 1} } },
    {$sort: {totalVendidos: -1}},
    {$limit : 1},
    {$project : { _id:0,CodBarras: 1, totalVendidos: 1, Nombre: 1}}
  ]
)
```

1.2.1. Ejemplo

```
db.articulos.insert( { CodBarras: 7231564345110546, Nombre: "1984",
  Compradores: [22222222] } )

db.articulos.insert( { CodBarras: 0231564105110546,
  Nombre: "El principito", Compradores: [333333333, 22222222] } )
```

1.3. Sectores donde trabaja exactamente 3 empleado.

1.3.1. Ejemplo

1.4. Empleado que trabaja en más sectores..

1.4.1. Ejemplo

1.5. Ranking de los clientes con mayor cantidad de compras.

1.5.1. Ejemplo

1.6. Cantidad de compras realizadas por clientes de misma edad.

1.6.1. Ejemplo

2. Map-Reduce.

Intro a MP, explicar como cargar datos y codigo.

2.1. Disposiciones de tipo resolucion en Abril de 2013.

IDEA: map: si el registro dado es de tipo resolucion y tiene fecha abril del 2013, emitir("resolucion",cant=1)

reduce: sumar los cant q nos pasan y emitir lo mismo ("resolucion",suma)

```
var map1 = function(){
  var date = this["FechaBOJA"].split('/')
  if(this["Tipo"] == "Resoluciones" && date[1]==4 && date[2]==2013){
    emit(this["Tipo"],1)
  }
}
```

```
var reduce1 = function(key,values){
  return Array.sum(values)
}
```

luego llamar a la función de map-reduce de la forma:

```
db.disposiciones.mapReduce(map1,reduce1,{out: parte2a})
```

Lo que nos devuelve:

```
{ "_id" : "Resoluciones", "value" : 607 }
```

2.2. Disposiciones de cada tipo.

2.3. Fecha mas citada.

2.4. Mayor cantidad de paginas por cada tipo.

3. Sharding.

4. Otras base de datos NoSql.

Base de datos key-value, usando el motor redis. Redis permite el uso de namespaces (tener varios "diccionarios", la terminología de redis para esto sería "multiple databases"), lo cual hace el diseño más prolijo y simple.

Usar map-reduce en redis no es algo built-in ni estandar, pero investigamos y es posible, por ejemplo, integrarlo con Hadoop.

Parte1: 1a: En redis hay un comando SCAN que permite iterar las claves. Con esto podemos iterar un diccionario de empleados, donde en cada value hay una lista de datos de cada cliente que atendió. Entonces se puede iterar las claves una por una y quedarse con las que tienen algun cliente mayor de edad. 1b: Usando SCAN se puede iterar las claves de un diccionario articulos -> ventas. Mediante codigo se buscan las claves que tengan |ventas| maximo.

1c: Idem usando SCAN. Si tenemos un diccionario sector -> [empleados], es cuestion de iterar y mediante código buscar cuando |empleados|==3

1d: Un diccionario empleado -> [sectores]. Idem 1c usando SCAN.

1e: Diccionario cliente -> [compras]. Idem 1c, pero ordenando mediante |compras|.

1f: Esta consulta ya es un poco más compleja, hay varios enfoques posibles. Uno es mantener las cosas simples y directamente usar un diccionario edad -> cantidadDeCompras. Por ser tan específico, es un diccionario que sólo sirve para esta consulta, con lo cual estamos agregando un costo de mantenimiento extra a la DB sólo por una query. Otra opción es valerse de map reduce y usar alguno de los otros diccionarios, como el de 1a. Como ventaja, la consulta es simple y no hace falta crear un diccionario extra solo por esta consulta.

Parte 2: Asumimos que se crea un id único para las disposiciones, podría ser un hash de sus datos o la combinación (numeroBoja, paginaInicial, PaginaFinal) que asumimos que identifica univocamente a la disposicion. Entonces se tiene un diccionario id -> disposicion.

Las resoluciones por map-reduce son prácticamente iguales a las versiones hechas en mongodb ya que la entrada de la función map es practicamente la misma.

Parte 3: TODO: primero habría que hacerlo en mongodb jaja.

5. Conclusiones.