Ayudantía Examen

Freddie Venegas A - fgvenegas@uc.cl

Temas

- XPath
- SPARQL
- SQL (yes, again)

XPath

```xml
<?xml version="1.0"?>
<Disqueras>
    <Disquera nombre = "Nombre Disquera 1">
        <Disco nombre = "Nombre disco 1" lanzamiento = "Año lanzamiento 1" >
            <CantidadCanciones>N° Canciones</CantidadCanciones>
            <Precio>Precio Disco</Precio>
            <Banda>Banda Disco</Banda>
        </Disco>
        <Disco nombre = "Nombre disco 2" lanzamiento = "Año lanzamiento 2" >
            ...
        </Disco>
        ...
    </Disquera>
    <Disquera nombre = "Nombre Disquera 2">
        ...
    </Disquera>
    ...
</Disqueras>
```

1. Todas las bandas.
2. Todos los discos de la banda 'Muse'.
3. Disqueras que venden más de dos discos.
4. Disqueras que venden solo un disco.
5. Disqueras que venden el disco 'Ok Computer'.

6. Disqueras que posean al menos un disco con un precio mayor a $20000.
7. Disqueras que posean al menos un disco con menos de 10 canciones.
8. Bandas que poseen un disco 'Best Of'.
9. Discos vendidos por al menos dos disqueras distintas.
10. Todos los precios de la disquera 'XMusicL'.
11. El disco con más canciones de la disquera 'SQLive'.
12. El disco más barato.
13. Disqueras que no venden el disco más barato.
14. Disquera que vende el disco 'Ok Computer' al menor precio.

SPARQL

RDF: Resource Description Framework

Ejemplo documento RDF:

```
@prefix entity: <http://entity.com/>
@prefix relation: <http://relationship.com/>
@prefix foaf: <http://xmlns.com/foaf/0.1/>

entity:JuanReutter foaf:name "Juan Reutter".
entity:CristianRiveros foaf:name "Cristian Riveros".
entity:JuanReutter relation:Dicta entity:BasesDeDatos.
entity:JuanReutter relation:Dicta entity:Logica.
entity:CristianRiveros relation:Dicta entity:MatDiscretas.
entity:JuanReutter foaf:friendOf entity:CristianRiveros.
entity:JuanReutter foaf:knows entity:CristianRiveros.
```

SPARQL: SPARQL Protocol And RDF Query Language

SPARQL

```
PREFIX entity: <http://entity.com/>
PREFIX relation: <http://relationship.com/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?x ?y
WHERE {
?x relation:Dicta entity:MatDiscretas.
?x foaf:name ?y.
}
```

x	y
<http://entity.com/CristianRiveros>	Cristian Riveros

SPARQL - FILTER

```
PREFIX entity: <http://entity.com/>
PREFIX relation: <http://relationship.com/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?p
WHERE {
?x relation:Dicta ?c.
?x ?p ?l.
FILTER isLiteral(?l)
}
```

p
<http://xmlns.com/foaf/0.1/name>

SPARQL - OPTIONAL

```
PREFIX entity: <http://entity.com/>
PREFIX relation: <http://relationship.com/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?x ?y
WHERE {
?x relation:Dicta ?c.
OPTIONAL{?x foaf:friendOf ?y }
}
```

x	y
<http://entity.com/JuanReutter>	<http://entity.com/CristianRiveros>
<http://entity.com/CristianRiveros>	

SPARQL - UNION

```
PREFIX entity: <http://entity.com/>
PREFIX relation: <http://relationship.com/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?c
WHERE {
{entity:JuanReutter relation:Dicta ?c.}
UNION
{entity:CristianRiveros relation:Dicta ?c.}
}
```

c
<http://entity.com/BasesDeDatos>
<http://entity.com/Logica>
<http://entity.com/MatDiscreta>

SPARQL - Ejercicios

Expandamos nuestro documento RDF, e incluyamos más personas y los cursos que estas realizaron. Existen ahora la relaciones: `relation:Cursa`, relation:SeDictaEn entity: `Semestre`. Escriba consultas para:

(a) Todas las personas que han dictado y cursado el mismo ramo, y si aparece, el semestre del ramo.

(b) Todas los alumnos de alguien llamado "Marcelo Arenas".

(c) Todos los ramos y la cantidad de personas que lo cursan.

Problema examen 2017-1

SQL

En esta pregunta, por simplicidad, asumimos que los tipos de elementos JSON son solo objetos y strings. En esta pregunta trabajaremos con una de las formas típicas de almacenar documentos JSON como un esquema relacional. La idea es asignar a cada valor un *id* distinto, y contar con dos tablas, una para los objetos y otra para los strings. A modo de ejemplo, abajo está la instancia que corresponde al almacenamiento de este JSON como una instancia relacional.

```
{
"profesores": {
    "p1": {"nombre" : "Denis", "apellido": "Parra"},
    "p2": {"nombre" : "Cristian", "apellido": "Riveros"},
    "p3": {"nombre" : "Marcelo", "apellido": "Arenas"}
    }
}
```

Objetos

id	key	value
0	"root"	1
1	"profesores"	2
1	"profesores"	3
1	"profesores"	4
2	"p1"	5
2	"p1"	6
3	"p2"	7
3	"p2"	8
4	"p3"	9
4	"p3"	10
5	"nombre"	11
6	"apellido"	12
7	"nombre"	13
8	"apellido"	14
9	"nombre"	15
10	"apellido"	16

Strings

id	value
11	"Denis"
12	"Parra"
13	"Cristian"
14	"Riveros"
15	"Marcelo"
16	"Arenas"

Más específicamente, notar que cada elemento es un objeto $\{k : v\}$ con llave k y valor v que puede ser o un string puro o un objeto compuesto de más pares. El esquema contiene una tabla *Objeto*, con atributos *id*, *key* y *value* y una tabla *String*, con atributos *id* y *value*. Para cada elemento $\{k : v\}$ en el documento JSON, se asigna un identificador (un *id*), y además se debe guardar (en el atributo *key*) la llave k del par y (en el atributo *value*) el id de los objetos en v, si v es en si mismo un objeto, o el id del string, si v es un string. Los id de objetos se almacenan en la tabla *Objetos*, mientras que los strings se almacenan en la tabla *Strings*. Notar que al documento completo se le asigna la llave "root" el identificador 0.

Parte a (3 ptos).
Escriba cómo obtener la información de estas consultas en MongoDB, pero usando SQL sobre la representación relacional de estos documentos que hemos explicado anteriormente.
(i) `db.entidades.find({}{"profesores.p1.apellido":1,"profesores.p1.nombre":1})`
(ii) `db.entidades.find({"profesores.p2.nombre":"Cristian"}{"profesores.p2.apellido":1})`

Parte b (3 ptos).
La representación relacional de documentos MongoDB con esta estructura es redundante. Explique dónde está la redundancia, y cómo se puede normalizar el esquema para eliminarla.