TPO - BD II 2024 1C

Grupo 7 - Integrantes

🧐 62533 - Liu, Jonathan

🐧 62494 - Wischñevsky, David

🦋 62495 - Vilamowski, Abril

Ejercicio 1 - MongoDB

a. Importe el archivo albumlist.csv (o su versión RAW) a una colección. Este archivo cuenta con el top 500 de álbumes musicales de todos los tiempos según la revista Rolling Stones.

Para importar los datos del archivo se utilizó el comando `mongoimport`

```
$ mongoimport --db ej1 --collection albums --type csv ./alb
umlist.csv --headerline
```

Luego se debe iniciar mongo, e indicar que se utilizará la base de datos `ej1`

```
$ mongosh
test> use('ej1')
```

b. Cuente la cantidad de álbumes por año y ordénelos de manera descendente

(mostrando los años con mayor cantidad de álbumes al principio).

Para contar los álbumes por año se utiliza la cláusula `aggregate` .

La agrupación se hace por el atributo `Year` , contando cuántos elementos tienen dicho atributo con ese valor.

Finalmente, se ordena de manera descendente.

```
ej1> db.albums.aggregate(
  { $group: {_id: "$Year", count: {$sum: 1} } },
  { $sort : { count : -1 } }
)
```

```
[
  { _id: 1970, count: 26 },
  { _id: 1972, count: 24 },
  { _id: 1973, count: 23 },
  { _id: 1969, count: 22 },
  { _id: 1968, count: 21 },
  { _id: 1971, count: 21 },
  { _id: 1967, count: 20 },
  { _id: 1977, count: 18 },
  { _id: 1975, count: 18 },
  { _id: 1978, count: 16 },
  { _id: 1994, count: 16 },
  { _id: 1965, count: 14 },
  { _id: 1979, count: 14 },
  { _id: 1974, count: 14 },
  { _id: 1966, count: 13 },
  { _id: 1991, count: 13 },
  { _id: 1987, count: 12 },
  { _id: 1976, count: 12 },
  { _id: 1985, count: 11 },
  { _id: 1984, count: 10 }
]
```

Notar que cada `_id` se corresponde con un año, y `count` es la cantidad de álbumes de dicho año.

c. A cada documento, agregarle un nuevo atributo llamado 'score' que sea 501 - *Number*.

A continuación se presentan dos maneras de agregar el atributo en cada documento:

1. La forma 1 es la manera declarativa, más convencional, que modifica a todos los documentos al mismo tiempo. Es más confiable y eficiente, pero menos versátil que la segunda y utiliza `$substract`.
2. La forma 2 es una manera más funcional, que busca uno por uno a todos los álbumes, y luego individualmente les agrega el atributo `score`. En resultado es equivalente a la forma 1, pero permite establecer directamente la fórmula para el score, y de ser necesario, permitiría agregar más lógica a las acciones realizadas sobre cada álbum.

```
// Forma 1:
ej1> db.albums.updateMany({}, [{ $set: { score: { $subtrac
t: [501, "$Number"] } } }])

// Forma 2:
ej1> db.albums.find({}).forEach((doc) => {
    db.albums.updateOne(
        { _id: doc._id },
        { $set: { score: 501 - Number(doc.Number) } }
    )
})
```

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 500,
  modifiedCount: 500,
  upsertedCount: 0
}
```

Resultado de la forma 1 (la 2 no produce salida):

d. Realice una consulta que muestre el 'score' de cada artista.

Para mostrar el `score` de cada artista, se ordenan los álbumes por `Artist` y se computa la suma de los scores individuales.

Para mayor claridad, se ordenó de manera descendente la suma de `score` .

```
ej1> db.albums.aggregate(
  { $group: { _id: "$Artist", score: { $sum: "$score" } }
},
  { $sort : { score : -1 } }
)
```

```
[
  { _id: 'The Beatles', score: 3855 },
  { _id: 'The Rolling Stones', score: 3604 },
  { _id: 'Bob Dylan', score: 3377 },
  { _id: 'Bruce Springsteen', score: 2251 },
  { _id: 'The Who', score: 2210 },
  { _id: 'Led Zeppelin', score: 2107 },
  { _id: 'Stevie Wonder', score: 1548 },
  { _id: 'Sly & The Family Stone', score: 1537 },
  { _id: 'Radiohead', score: 1509 },
  { _id: 'David Bowie', score: 1508 },
  { _id: 'U2', score: 1495 },
  { _id: 'The Byrds', score: 1353 },
  { _id: 'The Jimi Hendrix Experience', score: 1350 },
  { _id: 'Pink Floyd', score: 1316 },
  { _id: 'Elton John', score: 1303 },
  { _id: 'The Velvet Underground', score: 1272 },
  { _id: 'Various Artists', score: 1246 },
  { _id: 'Elvis Presley', score: 1246 },
  { _id: 'Michael Jackson', score: 1212 },
  { _id: 'Bob Marley & The Wailers', score: 1107 }
]
```

Notar que cada `_id` se corresponde con el nombre de un artista, y `score` es la suma de los scores de cada álbum de dicho artista.

Ejercicio 2 - Neo4J

a. ¿Cuántos productos hay en la base?

Se buscan todos los nodos de tipo Product, y se obtiene su cantidad

```
MATCH (n:Product) RETURN COUNT(*) AS CantProductos
```

```
╒═══════════════╕
│CantProductos│
╞═══════════════╡
│77             │
└───────────────┘
```

b. ¿Cuánto cuesta el “Queso Cabrales”?

Se buscan los productos con nombre “Queso Cabrales” y se obtienen sus precios unitarios (en este caso existe sólo uno con dicho nombre)

```
MATCH (n:Product {productName: "Queso Cabrales"})
RETURN n.unitPrice AS PrecioQuesoCabrales
```

```
╒═══════════════════╕
│PrecioQuesoCabrales│
╞═══════════════════╡
│21.0               │
└───────────────────┘
```

c. ¿Cuántos productos pertenecen a la categoría “Condiments”?

Un producto pertenece a una categoría si existe una arista de tipo `PART_OF` que va desde el producto a la categoría.

Se buscan productos que apunten a la categoría de nombre “Condiments”

```
MATCH (n:Product)-[:PART_OF]->(c:Category {categoryName: "C
ondiments"})
```

```
RETURN COUNT(n) AS CantProductosCondimentos
```

```
╒═══════════════════════════╕
│CantProductosCondimentos│
╞═══════════════════════════╡
│12                         │
└───────────────────────────┘
```

d. Del conjunto de productos que ofrecen los proveedores de “UK”, ¿Cuál es el nombre y el precio unitario de los tres productos más caros?

Analizando los datos, se descubrió que la información estaba mal cargada y, en muchos casos, tenía países y números de teléfono intercambiados.

En caso de que los datos fueran correctos, la consulta debería buscar los proveedores con país “UK” que tengan una arista que los conecte a productos, y retornar el nombre de esos productos, ordenando por precio de manera descendente.

Versión 1: Consulta que debería funcionar si la data estuviera bien cargada

```
MATCH (s:Supplier {country: "UK"})-[:SUPPLIES]->(p:Product)
RETURN p.productName AS Nombre, p.unitPrice AS PrecioUnitar
io
ORDER BY p.unitPrice DESC LIMIT 3
```

Resultado de la consulta inicial, no tiene sentido cortar en 3 si solo hay 3 productos de proveedores de UK

```
╒═══════════════╤═══════════════╕
│Nombre         │PrecioUnitario│
╞═══════════════╪═══════════════╡
│"Chang"        │19.0           │
├───────────────┼───────────────┤
│"Chai"         │18.0           │
```

```
├──────────────────┼──────────────────┤
|"Aniseed Syrup"|10.0           |
└──────────────────┴──────────────────┘
```

Debido a que los resultados eran poco satisfactorios (se buscaban los 3 de mayor precio, pero son los únicos 3 que hay), se planteo otra consulta, que considere que el país podía encontrarse en el número de teléfono.

Versión 2: Versión mejorada de la consulta

```
MATCH (s:Supplier)-[:SUPPLIES]->(p:Product)
WHERE s.phone = 'UK' OR s.country = "UK"
RETURN p.productName AS Nombre, p.unitPrice AS PrecioUnitar
io
ORDER BY p.unitPrice DESC LIMIT 3
```

También se planteo una consulta aún más abarcativa, que no solo considera teléfono y país, sino que genéricamente busca todos los proveedores con algún atributo igual a “UK”.

Versión 3: Versión abarcativa de la consulta

```
MATCH (s:Supplier)-[:SUPPLIES]->(p:Product)
WHERE any(attr IN keys(s) WHERE s[attr] = "UK")
RETURN p.productName AS Nombre, p.unitPrice AS PrecioUnitar
io
ORDER BY p.unitPrice DESC LIMIT 3
```

Resultado obtenido para las dos versiones que contemplan errores en la data

```
╒═══════════════════════════╤═══════════════╕
|Nombre                     |PrecioUnitario|
╞═══════════════════════════╪═══════════════╡
|"Sir Rodney's Marmalade"|81.0          |
```

```
├──────────────────────────┼──────────────────┤
│"Chang"                   │19.0              │
├──────────────────────────┼──────────────────┤
│"Chai"                    │18.0              │
└──────────────────────────┴──────────────────┘
```

Consideramos que sería preferible la versión 2, ya que la 3 podría ser demasiado abarcativa, y terminar seleccionando nodos de más o tener un impacto mayor en performance.

Ejercicio 3 - Redis

a. Importar los datos del archivo a Redis

Para importar los datos del archivo `bataxi.csv` se utilizó el siguiente comando en la terminal de linux.

```
awk -F, 'NR > 1 { system("redis-cli GEOADD bataxi " $6 " "
$7 " " $1) }' bataxi.csv
```

Esto procesa cada entrada del archivo, y para cada una corre el comando `redis-cli GEOADD bataxi` con los valores de las columnas `origen_viaje_x` , `origen_viaje_y` y `id_viaje_r` como argumentos.

Una vez importado el archivo, ejecutamos `redis-cli` .

b. ¿Cuántos viajes se generaron a 1 km de distancia de estos 3 lugares?

```
places = [{"place":"Parque Chas", "lon": -58.479258, "lat":
-34.582497},
{"place":"UTN" , "lon": -58.468606, "lat": -34.658304},
{"place":"ITBA Madero", "lon": -58.367862, "lat": -34.60293
8}];
```

Para cada lugar se ejecutó el siguiente comando dentro de `redis-cli`:

```
> GEOSEARCH bataxi
    FROMLONLAT @lon @lat
```

```
    BYRADIUS 1 km
    ASC
    WITHDIST
    WITHCOOR
```

Búsqueda de viajes que se realizaron dentro de un radio de 1 km desde un punto específico (definido por longitud y latitud). Los resultados se ordenan por distancia en orden ascendente, y se muestra tanto la distancia como las coordenadas de los lugares encontrados.

Donde se debe reemplazar `@lon` y `@lat` por la longitud y latitud del lugar.
Se le agregaron los argumentos `ASC`, `WITHDIST`, `WITHCOOR` para poder verificar que los viajes retornados efectivamente se hayan generado a 1 km de distancia.

Se obtienen los siguientes resultados:

- 339 resultados para *"Parque Chas"*
- 9 resultados para *"UTN"*
- 242 resultados para *"ITBA Madero"*

Aclaración: las consultas muestran todos los resultados numerados, de manera que la cantidad de resultados se obtuvo viendo el número del último viaje.

```
> GEOSEARCH bataxi
    FROMLONLAT -58.479258 -34.582497
    BYRADIUS 1 km
    ASC
    WITHDIST
    WITHCOORD
```

```
 1) 1) "17298"
    2) "0.1472"
    3) 1) "-58.47862333059310913"
       2) "-34.5812806205778287"
 2) 1) "17779"
    2) "0.1592"
    3) 1) "-58.48031312227249146"
```

```
       2) "-34.58135919693376792"
    ...

338) 1) "16188"
     2) "0.9772"
     3) 1) "-58.47866624593734741"
        2) "-34.57372461680184017"
339) 1) "16839"
     2) "0.9978"
     3) 1) "-58.4783819317817688"
        2) "-34.57355479048416669"
```

Ejemplo para "Parque Chas". Solo se muestran los primeros y últimos dos resultados

c. ¿Cuántas KEYS hay en la base de datos Redis?

```
> DBSIZE
1
```

Se ejecutó el comando `DBSIZE` y se obtuvo 1 de resultado, ya que fue la única key que se cargó en esa instancia de Redis.

d. ¿Cuántos miembros tiene la key 'bataxi'?

```
> ZCARD bataxi
19148
```

Se ejecutó el comando `ZCARD` para obtener la cantidad de miembros de `bataxi`, se utilizó este comando, pues el `GeoAdd` utiliza la estructura `zset` (como se verá en el siguiente item). Esto hace que no se pueda utilizar comandos para conjuntos no ordenados, por ejemplo `SCARD`.

Si se utiliza este comando se obtiene un error:

```
> SCARD bataxi
(error) WRONGTYPE Operation against a key holding the wrong k
```

e. ¿Sobre qué estructura de Redis trabaja el GeoADD?

```
> TYPE bataxi
zset
```

`GeoADD` utiliza la estructura `zset` , que es un conjunto ordenado. Esto permite el uso de comandos para conjuntos ordenados, generalmente prefijados con Z.

Algunos con su equivalente no ordenado, por ejemplo: `ZCARD`, `ZCOUNT`, `ZREM`, `ZSCAN`, etc.; que tienen su equivalente `SCARD`, `SCOUNT`, `SREM`, `SSCAN`, etc.

Además, existen comandos exclusivos a `zset` (sin equivalentes de conjuntos no ordenados). Por ejemplo: `ZPOPMIN`, `ZPOPMAX`, `ZRANK`, etc.