



Instituto Tecnológico de Buenos Aires
Base Datos II 72.41
Trabajo Práctico Obligatorio - 2024 1Q

Grupo 10

Integrantes:

Matilla, Juan Ignacio 60459

jmatilla@itba.edu.ar

Burgos Sosa, Jose Leon 61525

jburos@itba.edu.ar

Curti, Pedro 61616

pcurti@itba.edu.ar

Repositorio: github.com/pcurti/TPO-DB2

- hash: b93533cde6fb4fd6461ff02fa7150aded414e710

Índice

Ejercicio 1 - MongoDB.....	2
Resumen.....	2
Enunciados.....	2
Ejecución.....	2
Resolución.....	3
Endpoints.....	3
Ejercicio 2 - Neo4J.....	5
Resumen.....	5
a) ¿Cuántos productos hay en la base?.....	5
b) ¿Cuánto cuesta el “Queso Cabrales”?.....	6
c) ¿Cuántos productos pertenecen a la categoría “Condiments”?.....	7
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?.....	8
Ejercicio 3 - Redis.....	10
Resumen.....	10
a) Importar los datos del archivo a Redis.....	10
b) ¿Cuántos viajes se generaron a 1 km de distancia de estos 3 lugares?.....	11
c) ¿Cuántas KEYS hay en la base de datos Redis?.....	12
d) ¿Cuántos miembros tiene la key 'bataxi'?.....	12
e) ¿Sobre qué estructura de Redis trabaja el GeoADD?.....	12

Ejercicio 1 - MongoDB

Resumen

Para la resolución de los siguientes ejercicios se implementó una **API** utilizando **node.js** la cual contiene 5 endpoints uno para la resolución de cada inciso y uno extra para borrar los datos de la colección.

Además de node.js se utilizaron dependencias como:

- express.
- dotenv.
- mongodb.
- csv-parser.

Enunciados

- 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.
- Cuente la cantidad de álbumes por año y ordénelos de manera descendente (mostrando los años con mayor cantidad de álbumes al principio).
- A cada documento, agregarle un nuevo atributo llamado 'score' que sea *501 - Number*.
- Realice una consulta que muestre el 'score' de cada artista.

Ejecución

Para poder ejecutar la **API**, utilizaremos [codespace de github](#), una vez iniciado utilizaremos los siguientes comando de bash:

1. docker-compose build.
2. docker-compose up app.

Esta API levanta una WEB en localhost en el puerto 3000 la cual brindara 5 links que ejecutarán los endpoints. (<http://www.localhost:3000/>)

Resolución

Endpoints

a) */load-files*

```
...
fs.createReadStream(csvFilePath)
  .pipe(csv())
  .on('data', (data) => results.push(data))
  .on('end', async () => {
    try {
      await collection.insertMany(results);
      console.log(`${results.length} documents inserted successfully`);
      res.status(200).send(`<h1>${results.length} Files loaded successfully</h1>`);
    } catch (err) {
      console.error('Error inserting documents into MongoDB:', err);
      res.status(500).send('Internal Server Error');
    } finally {
      await client.close();
    }
  })
...

```

b) */count-albums-by-year*

```
...
const result = await collection.aggregate([
  {
    $group: {
      _id: "$Year",
      count: { $sum: 1 }
    }
  },
  {
    $sort: { count: -1 }
  }
]).toArray();
...

```

c) */add-score-attribute*

```
...
const result = await collection.updateMany({}, [{ $set: { score: { $subtract: [501, { $toInt: "$Number"
}} } } }]);
...
```

d) */query-artist-score*

Nota: El score del artista es la suma de todos los scores de sus álbumes dividido por la cantidad de álbumes que le pertenece.

```
...
const result = await collection.aggregate([
  {
    $group: {
      _id: "$Artist",
      totalScore: { $sum: "$score" },
      albumCount: { $sum: 1 }
    }
  },
  {
    $project: {
      _id: 0,
      Artist: "$_id",
      averageScore: { $divide: ["$totalScore", "$albumCount"] }
    }
  }
]).toArray();
...
```

e) */drop-collection*

```
...
await db.collection(process.env.COLLECTION).drop();
...
```

Ejercicio 2 - Neo4J

Resumen

Para la resolución de los siguientes ejercicios se utilizó el Sandbox de Neo4J mencionado en el enunciado y se sacaron capturas de pantalla para mostrar los resultados como se indicó en la clase de presentación del trabajo práctico.

Citas de la documentación oficial utilizada en las explicaciones:

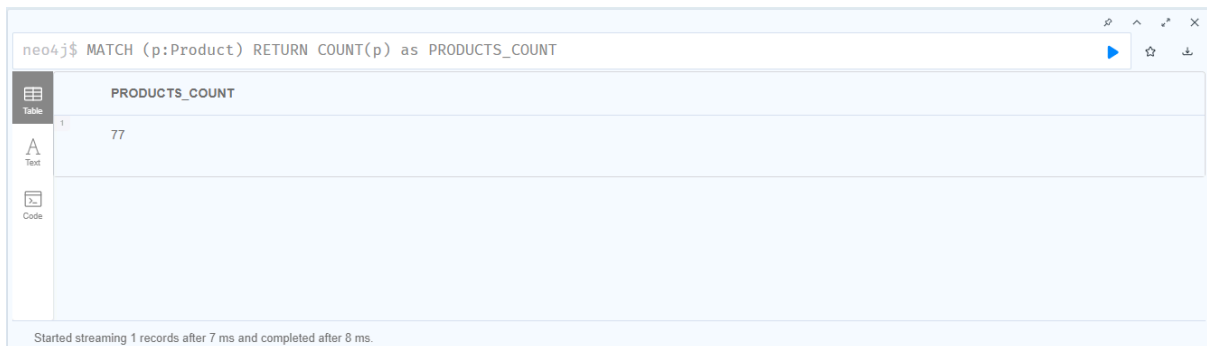
<https://neo4j.com/docs/cypher-manual/current/queries/basic/>

a) ¿Cuántos productos hay en la base?

Query:

```
MATCH (p:Product) RETURN COUNT(p) as PRODCUTS_COUNT
```

Captura del sandbox de Neo4J:



The screenshot shows the Neo4J Sandbox interface. At the top, the query `neo4j$ MATCH (p:Product) RETURN COUNT(p) as PRODCUTS_COUNT` is entered in the input field. Below the input field, there are three tabs: 'Table', 'Text', and 'Code'. The 'Table' tab is selected, displaying a table with one column named 'PRODUCTS_COUNT' and one row containing the value '77'. At the bottom of the interface, a status message reads: 'Started streaming 1 records after 7 ms and completed after 8 ms.'

PRODUCTS_COUNT
77

Utilizando *MATCH(p:Product)* indico que quiero buscar los nodos de productos luego mediante la función de agregación *COUNT(p)* se contabilizan las filas que coinciden con la consulta, en este caso, todas aquellas que pertenecen al nodo Product. Utilizando 'as', se renombra el resultado para que su denominación sea más clara. El resultado es que hay 77 productos en total.

b) ¿Cuánto cuesta el “Queso Cabrales”?

Query:

```
MATCH (p:Product{productName: 'Queso Cabrales'}) RETURN p
```

Captura del sandbox de Neo4J:



The screenshot shows the Neo4J sandbox interface. At the top, the query editor contains the Cypher query: `neo4j$ MATCH (p:Product{productName: 'Queso Cabrales'}) RETURN p`. Below the editor, the results are displayed in a table view. The table has one column labeled 'p' and one row containing a JSON object representing the product. The JSON object includes properties such as 'identity', 'labels', 'properties', 'reorderLevel', 'unitPrice', 'unitsInStock', 'supplierID', 'productID', 'quantityPerUnit', 'discontinued', 'productName', 'unitsOnOrder', and 'categoryID'. The status bar at the bottom indicates 'Started streaming 1 records after 6 ms and completed after 7 ms.'

Utilizando `p:Product {productName: 'Queso Cabrales'}`, se especifica una condición para el nodo buscado: su atributo `productName` debe ser igual a 'Queso Cabrales'. Esta sintaxis, tal como se presenta en la documentación oficial, permite filtrar los nodos según atributos específicos. Una vez identificado el nodo que cumple con este criterio, el comando `RETURN` se utiliza para devolver el o los nodos que cumplan la condición.

Cita de la documentación oficial:

§ Finding nodes

The `MATCH` clause is used to find a specific pattern in the graph, such as a specific node. The `RETURN` clause specifies what of the found graph pattern to return.

For example, this query will find the nodes with `Person` label and the name `Keanu Reeves`, and return the `name` and `born` properties of the found nodes:

Query

```
MATCH (keanu:Person {name:'Keanu Reeves'})
RETURN keanu.name AS name, keanu.born AS born
```

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

Query:

```
MATCH (c:Category {categoryName: 'Condiments'})<-[r]-(p:Product)
RETURN COUNT(p) as CONDIMENTS_PRODUCTS_COUNT
```

Captura del sandbox de Neo4J:

The screenshot shows the Neo4J sandbox interface. At the top, a query editor contains the following Cypher query:

```
1 MATCH (c:Category {categoryName: 'Condiments'})<-[r]-(p:Product)
2 RETURN COUNT(p) as CONDIMENTS_PRODUCTS_COUNT
```

Below the query editor, the results are displayed in a table view. The table has a single column header `CONDIMENTS_PRODUCTS_COUNT` and one data row with the value `12`.

CONDIMENTS_PRODUCTS_COUNT
12

At the bottom of the interface, a status message reads: "Started streaming 1 records after 8 ms and completed after 9 ms."

Esta consulta restringe la búsqueda a aquellos cuyo atributo `categoryName` es igual a `'Condiments'`. Luego, establece una relación de salida desde cualquier nodo `Product` hacia el nodo `Category` especificado mediante el uso de `<-[r]`. Como se detalla en la documentación oficial, al utilizar `[r]` en la relación, se busca por el atributo que une los nodos, lo cual en este caso conecta productos con su categoría correspondiente.

Finalmente, el comando `RETURN` se emplea junto con la función de agregación `COUNT(p)` para contar el número de nodos `Product` que están relacionados con el nodo `Category` especificado. El resultado de esta operación se devuelve con el alias `CONDIMENTS_PRODUCTS_COUNT`, proporcionando un nombre claro de la cantidad de productos categorizados como `'Condiments'`.

Cita de la documentación oficial:

It also possible to look for the type of relationships that connect nodes to one another. The below query searches the graph for outgoing relationships from the `Tom Hanks` node to any `Movie` nodes, and returns the relationships and the titles of the movies connected to him.

Query

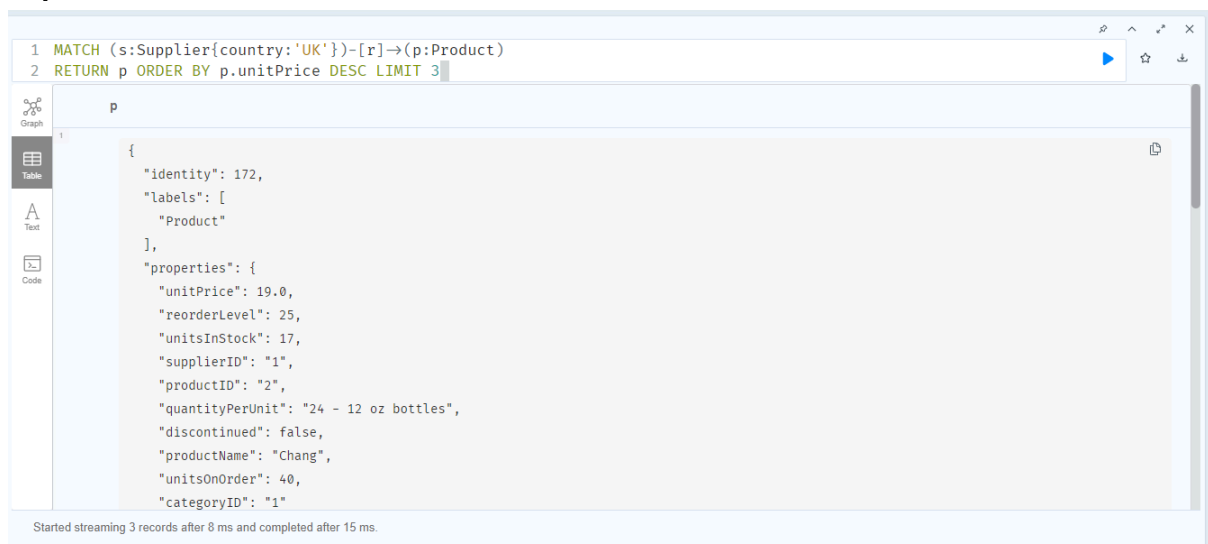
```
MATCH (tom:Person {name:'Tom Hanks'})-[r]->(m:Movie)
RETURN type(r) AS type, m.title AS movie
```


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?

Query:

```
MATCH (s:Supplier{country:'UK'})-[r]->(p:Product)
RETURN p ORDER BY p.unitPrice DESC LIMIT 3
```

Captura del sandbox de Neo4J:



Esta consulta emplea el comando *MATCH* para localizar nodos etiquetados como *Supplier* dentro de la base de datos, restringiendo la búsqueda a aquellos cuyo atributo *country* es igual a 'UK'. Se establece una relación de salida desde estos nodos *Supplier* hacia nodos *Product* utilizando *-[r]->*. Como fue mencionado anteriormente, al utilizar *[r]* en la relación, se busca por el atributo que une los nodos, conectando proveedores con los productos que suministran.

La consulta luego emplea *RETURN* para devolver los atributos *productName* y *unitPrice* de los nodos *Product* encontrados. Para ordenar los resultados por el precio de los productos, se utiliza *ORDER BY p.unitPrice DESC*, que ordena los productos de manera descendente, es decir, del más caro al más barato. Finalmente, al usar *LIMIT 3*, se restringe el resultado a solo los tres productos más caros. Este proceso permite extraer y visualizar los nombres y precios unitarios de los tres productos más costosos suministrados por proveedores del Reino Unido.

Ejercicio 3 - Redis

Resumen

Para la resolución de los ejercicios que están a continuación se creó un script de Python que aprovecha la API ofrecida por Redis. Las respuestas incluyen los fragmentos de código de la misma y un link a la documentación oficial de cada función que ofrece Redis.

a) Importar los datos del archivo a Redis

```
# Abre el archivo csv en modo lectura
with open(file_path, 'r') as file:
    # Crea un cursor que recorre el archivo CSV línea por línea
    reader = csv.reader(file)
    # Saltea la primer línea ya que es la de los títulos
    next(reader)
    # Abre la conexión a redis
    with redis.Redis(host='localhost', port=6379, decode_responses=True) as r:
        count = 0
        print('====IMPORTING====')
        # Loopea las líneas del CSV
        for row in reader:
            lon = float(row[origen_viaje_x])
            lat = float(row[origen_viaje_y])
            member = row[id_viaje_r]
            # Llama a GEOADD y pasa los argumentos
            count += r.geoadd(KEY, [lon, lat, member])
        print('Added', count, 'members to ', KEY)
        ...
```

En esta solución se itera el archivo línea por línea y se llama a la función [GEOADD](#) a través de la API. En esta se pasan los argumentos como los solicitó la cátedra, como se ve a continuación:

- key → "bataxi"
- longitude → origen_viaje_x
- latitude → origen_viaje_y
- member → id_viaje_r

Al final se imprime el total de datos añadidos a la base de datos. Cabe destacar que si el member ya era parte del keyspace, teniendo en cuenta que no se ha puesto ninguna flag, el call a GEOADD retornara 0.

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

```
...
places = {
    'Parque Chas': { 'lon' : -58.479258, 'lat': -34.582497},
    'UTN' : { 'lon' : -58.468606, 'lat': -34.658304},
    'ITBA Madero': { 'lon' : -58.367862, 'lat': -34.602938}
}

for place in places:
    lon = places[place]['lon']
    lat = places[place]['lat']
    response = r.geosearch(
        KEY,
        longitude=lon,
        latitude=lat,
        radius=1,
        unit="km"
    )
    print('Trips near ', place, ': ', len(response))
...
```

Para solucionar este ejercicio utilizamos la función [GEOSEARCH](#). La misma nos trae un array con aquellos members que se encuentran dentro del radio de búsqueda. Ofrece distintos sistemas de medición, y la posibilidad de hacer áreas rectangulares de búsqueda. En este caso usamos el radio en 1 km. Luego aplicamos **len** de la respuesta para saber cuántos miembros tiene el array que es lo que nos pide el enunciado.

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

```
...
print('Keyspace info -> ammount of keys:', r.info('keyspace')['db0']['keys'])
...
```

En este caso utilizamos la función [INFO](#). Esta devuelve información y estadísticas de la base de datos. En este caso además aclaramos la sección 'keyspace' que es la que nos brinda las estadísticas sobre las keys de nuestra base de datos.

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

```
...
print('Members in ', KEY, ': ', r.zcard(KEY))
...
```

Para este punto utilizamos [ZCARD](#) que nos permite obtener la cardinalidad de los sorted sets.

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

Podemos ver lo que dice la documentación oficial en la sección de [GEOADD](#), que nos da una pista sobre como redis trata a sus índices GEOESPACIALES:

Note: there is no GEODEL command because you can use [ZREM](#) to remove elements. The Geo index structure is just a sorted set.

Como podemos ver se trata de un set ordenado.