

UNIR - Universidad Internacional de la Rioja

Ciudad de México

ACTIVIDAD 2:

Uso avanzado de bases de datos NoSQL

Alumnos:

Paola Michelle Figueroa Benitez

Lowenski Paredes Rosario

Carlos Damian Rodriguez Uitzil

Cuenca Roa Leonard Jose

Grupo: 1001

Equipo 01C

ÍNDICE

Prepara el entorno de trabajo	3
Inicia la monitorización en MongoDB	3
Importar la base de datos	4
Exportar la base de datos	12
Restaurar una base de datos	13
Caso de uso	15
Ejercicio #1:	18
Ejercicio #3:	20
Ejercicio #4:	20
Ejercicio #5:	22
Ejercicio #6:	22
Ejercicio #7:	23
Ejercicio #8:	24
Ejercicio #9:	25
Ejercicio #10:	26
Grafos	27
Explicación paso a paso	27
Modelo	30
Import y Visualización del Grafo	31
Consulta CQL	31
Explicación detallada de cada parte del query:	32
Retorno y visualización:	33
Para mayor visualización del elemento grafo	33

Prepara el entorno de trabajo

Se implementó un entorno de desarrollo local consistente en MongoDB versión 8.0.4 y Mongosh 2.3.6 en cada equipo de los participantes. Esta configuración permitió a cada miembro del equipo trabajar de manera independiente y sincronizada, facilitando la colaboración y el testing.

```
➜ /opt/homebrew/var/log/mongodb on master at 14:15:23
> mongosh
Current Mongosh Log ID: 676db980808ebe76abaf1d69
Connecting to: mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.3.6
Using MongoDB: 8.0.4
Using Mongosh: 2.3.6

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

The server generated these startup warnings when booting
2024-12-26T14:14:46.210-06:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
2024-12-26T14:14:46.210-06:00: You are running this process as the root user, which is not recommended
```

Inicia la monitorización en MongoDB

Para llevar a cabo esta actividad, se implementó una estrategia de ejecución simultánea en tres terminales. En la primera, se utilizó el demonio de MongoDB (mongod) para crear la base de datos 'inspections' y 'countries' y las colecciones 'act-grupal-city_inspections', 'act-grupal-countries-big' y 'act-grupal-countries-small'. Esta configuración inicial se realizó directamente desde la línea de comandos de MongoDB.

Con el objetivo de monitorear el rendimiento y la salud de la base de datos durante el proceso, se abrieron dos terminales adicionales. En estas, se ejecutaron las herramientas de monitoreo mongotop y mongostat. El comando 'mongotop 10' proporcionó una vista en tiempo real de las operaciones más costosas en la base de datos, mientras que 'mongostat --discover' ofreció un resumen general del estado del servidor y las estadísticas de las bases de datos.

Creación de Base Datos y sus colecciones

```
// Crear la base de datos
use inspections;
// crear colecciones
db.createCollection('act-grupal-city_inspections')
// visualizar lo que se generó
show collections
//Otra forma de ver
db.getCollectionNames()
```

```
inspections> show collections
act-grupal-city_inspections
inspections> |
```

Creación de Base Datos y sus colecciones

```
// Crear la base de datos
use countries;
// crear colecciones
db.createCollection('act-grupal-countries-big')
db.createCollection('act-grupal-countries-small')
// visualizar lo que se generó
show collections
//Otra forma de ver
db.getCollectionNames()
```

```
inspections> use countries
switched to db countries
countries> db.createCollection('act-grupal-countries-big')
{ ok: 1 }
countries> db.createCollection('act-grupal-countries-small')
{ ok: 1 }
countries> show collections
act-grupal-countries-big
act-grupal-countries-small
countries> |
```

Importar la base de datos

Ahora para monitoreo de la carga de los datos la actividad número dos, nos proporciono los ficheros que contiene los datos json para las colecciones creadas se procedió a cargar, validar y analizar los resultados que arroja las terminales de monitoreo, Para esta tarea de

importación de datos, se aprovechó la rapidez y facilidad que ofrece Compass, una excelente herramienta de administración de bases de datos MongoDB. Previamente, se configuró Compass para utilizar la base de datos instalada localmente

Importar datos

Se importó los datos en la BD llamada **inspections**,

validando que en efecto esté el total de datos 81047.

The screenshot displays the MongoDB Compass application. On the left, the 'My Queries' section shows a list of connections: 'client1@localhost:27017' (selected), 'admin', 'config', 'countries', 'expeditions', 'local', and 'test-group-countries-test'. The main area shows the 'LOCAL' database with the 'inspections' collection selected. A document is highlighted with the following fields:

```

_id: ObjectId("56d1033179ec4de48d0524")
date: "2013-01-23T00:00:00Z"
business_name: "KELLOGG'S OLD GROCERY INC."
sector: "Cigarette Retail Dealer - 121"
address: "101 Main Street"
...
```

```

Below the main interface is a terminal window titled 'MongoDB' showing the raw MongoDB shell code for the same document:

```

> db.inspections.findOne()
{
 "_id": ObjectId("56d1033179ec4de48d0524"),
 "date": "2013-01-23T00:00:00Z",
 "business_name": "KELLOGG'S OLD GROCERY INC.",
 "sector": "Cigarette Retail Dealer - 121",
 "address": "101 Main Street"
}
```

```

Importar datos

Se importó los datos en la BD llamada **countries**, validando que en efecto esté el total de datos 21640 para la colección **act-grupal-countries-big**.

MongoDB Compass - LOCAL\countries.act-grupal-countries-big

Documents 21640 Aggregations Schema Indexes Validation

ADD DATA EXPORT DATA UPDATE DELETE

Import completed. 1 document imported.

MongoDB Compass - LOCAL\countries

Documents 21640 Aggregations Schema Indexes Validation

ADD DATA EXPORT DATA UPDATE DELETE

Import completed. 1 document imported.

Importar datos

Se importó los datos en la BD llamada **countries**, validando que en efecto esté el total de 1 para la colección **act-grupal-countries-small**.

MongoDB Compass - LOCAL\countries.act-grupal-countries-small

Documents 1 Aggregations Schema Indexes Validation

ADD DATA EXPORT DATA UPDATE DELETE

Import completed. 1 document imported.

MongoDB Compass - LOCAL\countries

Documents 1 Aggregations Schema Indexes Validation

ADD DATA EXPORT DATA UPDATE DELETE

Import completed. 1 document imported.

Análisis del comando mongotop al importar “act-grupal-city_inspections.json”

Al momento de importar los datos, se nos presenta un proceso dividido en tres fases que reflejan el tiempo total empleado, el cual es la suma del tiempo invertido en las operaciones de lectura y escritura realizadas por el procesador en ese instante preciso.

Primera fase: En esta etapa, se utilizó la operación de lectura durante 37 milisegundos.

Segunda fase: Continuando con el proceso, se mantuvo el uso de la operación de lectura, esta vez durante 68 milisegundos.

Tercera fase: En esta fase, se observa un cambio en la operación, pasando a utilizar la operación de escritura, la cual consumió 560 milisegundos.

Análisis:

A partir de los datos obtenidos, podemos concluir que las operaciones de escritura son más complejas que las de lectura, lo que conlleva un mayor tiempo de procesamiento por parte del procesador. Asimismo, se puede deducir que, durante el proceso de importación, la herramienta realiza una lectura completa del documento JSON antes de proceder a escribir los datos en la base de datos denominada 'inspections'

```
mongotop 10

      ns      total    read   write  2024-12-26T16:16:02-06:00
countries.act-grupal-countries-big  37ms  37ms  0ms
countries.act-grupal-countries-small 37ms  37ms  0ms
  admin.$cmd.aggregate  0ms  0ms  0ms
  admin.atlascli  0ms  0ms  0ms
  admin.system.version 0ms  0ms  0ms
  config.collections  0ms  0ms  0ms
  config.system.sessions 0ms  0ms  0ms
  config.transactions 0ms  0ms  0ms
inspections.act-grupal-city_inspections 0ms  0ms  0ms
  local.system.replset 0ms  0ms  0ms

      ns      total    read   write  2024-12-26T16:16:12-06:00
inspections.act-grupal-city_inspections 68ms  68ms  0ms
  config.collections 1ms  1ms  0ms
  admin.$cmd.aggregate 0ms  0ms  0ms
  admin.atlascli  0ms  0ms  0ms
  admin.system.version 0ms  0ms  0ms
  config.system.sessions 0ms  0ms  0ms
  config.transactions 0ms  0ms  0ms
countries.act-grupal-countries-big  0ms  0ms  0ms
countries.act-grupal-countries-small 0ms  0ms  0ms
  local.system.replset 0ms  0ms  0ms

      ns      total    read   write  2024-12-26T16:16:22-06:00
inspections.act-grupal-city_inspections 560ms  0ms  560ms
  admin.$cmd.aggregate 0ms  0ms  0ms
  admin.atlascli  0ms  0ms  0ms
  admin.system.version 0ms  0ms  0ms
  config.collections 0ms  0ms  0ms
  config.system.sessions 0ms  0ms  0ms
  config.transactions 0ms  0ms  0ms
countries.act-grupal-countries-big  0ms  0ms  0ms
countries.act-grupal-countries-small 0ms  0ms  0ms
  local.system.replset 0ms  0ms  0ms

      ns      total    read   write  2024-12-26T16:16:32-06:00
inspections.act-grupal-city_inspections 11ms  4ms  7ms
  admin.$cmd.aggregate 0ms  0ms  0ms
  admin.atlascli  0ms  0ms  0ms
  admin.system.version 0ms  0ms  0ms
  config.collections 0ms  0ms  0ms
  config.system.sessions 0ms  0ms  0ms
```

Análisis del comando mongostat --discover al importar “act-grupal-city_inspections.json”

Este tipo de herramienta nos proporciona en tiempo real la cantidad de operaciones que se realizan por segundo, para el caso de importar la colección nos arroja que se realizó un conjunto de inserciones sobre la colección mencionada, durante la instancia de Mongo DB en local nos arroja que se realizaron mas de 21.000 mil operaciones durante un segundo terminando el proceso de inserción en 4 bloques de procesos registrados usando un promedio de memoria de 0.725%.

mongostat --discover																		
localhost	*0	*0	*0	*0	0	3 0	0.0%	0.0%	0	393G	45.0M	0 0	0 0	2,15k	87.9k	29 Dec 26 16:16:02.576		
localhost	*0	*0	*0	*0	0	3 0	0.0%	0.0%	0	393G	45.0M	0 0	0 0	312b	83.7k	29 Dec 26 16:16:03.575		
localhost	*0	*0	*0	*0	0	0 0	0.0%	0.0%	0	393G	45.0M	0 0	0 0	111b	82.8k	29 Dec 26 16:16:04.578		
localhost	*0	*0	*0	*0	0	2 0	0.0%	0.0%	0	393G	43.0M	0 0	0 0	246b	83.5k	29 Dec 26 16:16:05.576		
localhost	*0	*0	*0	*0	0	0 0	0.0%	0.0%	0	393G	44.0M	0 0	0 0	111b	82.8k	29 Dec 26 16:16:06.578		
localhost	*0	*0	*0	*0	0	1 0	0.0%	0.0%	0	393G	44.0M	0 0	0 0	112b	83.3k	29 Dec 26 16:16:07.575		
localhost	*0	*0	*0	*0	0	1 0	0.0%	0.0%	0	393G	44.0M	0 0	0 0	173b	83.2k	29 Dec 26 16:16:08.577		
host insert query update delete getmore command dirty used flushes vszie res qrw arw net_in net_out conn time																		
localhost	*0	*0	*0	*0	0	1 0	0.0%	0.0%	0	393G	44.0M	0 0	0 0	245b	83.1k	29 Dec 26 16:16:09.579		
localhost	*0	*0	*0	*0	0	0 0	0.0%	0.0%	0	393G	44.0M	0 0	0 0	111b	83.0k	29 Dec 26 16:16:10.579		
localhost	*0	*0	*0	*0	0	6 0	0.0%	0.0%	0	393G	44.0M	0 0	0 0	560b	84.7k	29 Dec 26 16:16:11.578		
localhost	*0	*0	*0	*0	0	2 0	0.0%	0.0%	0	393G	44.0M	0 0	0 0	198b	86.7k	29 Dec 26 16:16:12.576		
localhost	*0	*0	*0	*0	0	2 0	0.0%	0.0%	0	393G	44.0M	0 0	0 0	311b	83.6k	29 Dec 26 16:16:13.577		
localhost	*0	*0	*0	*0	0	1 0	0.0%	0.0%	0	393G	44.0M	0 0	0 0	112b	83.1k	29 Dec 26 16:16:14.576		
localhost	*0	*0	*0	*0	0	1 0	0.0%	0.0%	0	393G	44.0M	0 0	0 0	245b	83.1k	29 Dec 26 16:16:15.578		
localhost	*0	*0	*0	*0	0	0 0	0.0%	0.0%	0	393G	44.0M	0 0	0 0	111b	82.9k	29 Dec 26 16:16:16.580		
localhost	*0	*0	*0	*0	0	2 0	0.0%	0.0%	0	393G	44.0M	0 0	0 0	176b	84.5k	29 Dec 26 16:16:18.577		
localhost no data received host insert query update delete getmore command dirty used flushes vszie res qrw arw net_in net_out conn time																		
localhost	21890	*0	*0	*0	0	1 0	0.3%	0.3%	0	393G	59.0M	0 0	0 0	6,17m	84.9k	29 Dec 26 16:16:19.575		
localhost	22936	*0	*0	*0	0	0 0	0.6%	0.6%	0	393G	72.0M	0 0	0 0	6,48m	84.9k	29 Dec 26 16:16:20.577		
localhost	24082	*0	*0	*0	0	6 0	0.9%	0.9%	0	393G	86.0M	0 0	0 0	6,80m	87.1k	29 Dec 26 16:16:21.574		
localhost	11991	4	*0	*0	0	3 0	1.1%	1.1%	0	393G	93.0M	0 0	0 0	3,38m	99.2k	29 Dec 26 16:16:22.579		
localhost	*0	*0	*0	*0	0	3 0	1.1%	1.1%	0	393G	93.0M	0 0	0 0	312b	83.8k	29 Dec 26 16:16:23.577		
localhost	*0	*0	*0	*0	0	0 0	1.1%	1.1%	0	393G	93.0M	0 0	0 0	111b	82.8k	29 Dec 26 16:16:24.579		
localhost	*0	*0	*0	*0	0	2 0	1.1%	1.1%	0	393G	66.0M	0 0	0 0	247b	83.7k	29 Dec 26 16:16:25.575		
localhost	*0	*0	*0	*0	0	1 0	1.1%	1.1%	0	393G	56.0M	0 0	0 0	112b	83.2k	29 Dec 26 16:16:26.572		
localhost	*0	*0	*0	*0	0	0 0	1.1%	1.1%	0	393G	55.0M	0 0	0 0	111b	83.0k	29 Dec 26 16:16:27.572		
localhost	*0	*0	*0	*0	0	1 0	1.1%	1.1%	0	393G	54.0M	0 0	0 0	173b	82.9k	29 Dec 26 16:16:28.577		

Análisis del comando mongotop al importar “act-grupal-countries-big.json”

Al momento de importar los datos, se nos presenta un proceso en una sola fase que reflejan el tiempo total empleado, el cual es la suma del tiempo invertido en las operaciones de lectura y escritura realizadas por el procesador en ese instante preciso.

Primera fase: En esta etapa, se utilizó la operación de lectura durante 246 milisegundos y en segundo plano las operaciones de escritura generando un tiempo de 117 milisegundos con un total de la operación de 363 milisegundos.

Análisis:

A partir de los datos obtenidos, podemos concluir que las operaciones de lectura son más complejas que las de escritura, lo que conlleva un mayor tiempo de procesamiento por parte del procesador. Asimismo, se puede deducir que, durante el proceso de importación, la herramienta realiza una lectura completa del documento JSON antes de proceder a escribir los datos en la base de datos denominada ‘countries’.

```
mongotop 10
  ns      total    read   write   2024-12-26T17:18:05-06:00
  admin.$cmd.aggregate   0ms   0ms   0ms
  admin.atlascli        0ms   0ms   0ms
  admin.system.version  0ms   0ms   0ms
  config.collections    0ms   0ms   0ms
  config.system.sessions 0ms   0ms   0ms
  config.transactions   0ms   0ms   0ms
  countries.act-grupal-countries-big 0ms   0ms   0ms
  countries.act-grupal-countries-small 0ms   0ms   0ms
  inspections.act-grupal-city_inspections 0ms   0ms   0ms
  local.system.replset  0ms   0ms   0ms

  ns      total    read   write   2024-12-26T17:18:13-06:00
countries.act-grupal-countries-big 363ms  246ms  117ms
  config.collections   1ms   1ms   0ms
  admin.$cmd.aggregate 0ms   0ms   0ms
  admin.atlascli        0ms   0ms   0ms
  admin.system.version  0ms   0ms   0ms
  config.system.sessions 0ms   0ms   0ms
  config.transactions   0ms   0ms   0ms
  countries.act-grupal-countries-small 0ms   0ms   0ms
  inspections.act-grupal-city_inspections 0ms   0ms   0ms
  local.system.replset 0ms   0ms   0ms
```

Análisis del comando mongostat --discover al importar “act-grupal-countries-big.json”

Esta herramienta nos proporciona un seguimiento instantáneo del número de operaciones que se ejecutan por segundo en la base de datos. Durante la importación de la colección, se observó un alto volumen de inserciones, superando las 18,000 operaciones por segundo en la instancia local de MongoDB. El proceso se finalizó en dos fases distintas, consumiendo un promedio de solo 0.1% de la memoria disponible.

mongostat --discover																		
localhost	*0	*0	*0	*0	0	1 0	0.0%	1.1%	0	393G 36.0M 0 0 0 0	299b	83.5k	35	Dec 26	17:17:52.625			
localhost	*0	*0	*0	*0	0	3 0	0.0%	1.1%	0	393G 37.0M 0 0 0 0	301b	87.1k	35	Dec 26	17:17:53.625			
localhost	*0	*0	*0	*0	0	2 0	0.0%	1.1%	0	393G 37.0M 0 0 0 0	246b	83.5k	35	Dec 26	17:17:54.623			
localhost	*0	*0	*0	*0	0	1 0	0.0%	1.1%	0	393G 36.0M 0 0 0 0	112b	83.3k	35	Dec 26	17:17:56.614			
localhost	no data received																	
localhost	*0	*0	*0	*0	0	0 0	0.0%	1.1%	0	393G 36.0M 0 0 0 0	110b	82.0k	35	Dec 26	17:17:57.626			
localhost	*0	*0	*0	*0	0	2 0	0.0%	1.1%	0	393G 37.0M 0 0 0 0	303b	82.3k	35	Dec 26	17:17:58.642			
	host insert query update delete getmore command dirty used flushes vsize res qrw arw net_in net_out conn time																	
localhost	*0	*0	*0	*0	0	1 0	0.0%	1.1%	0	393G 37.0M 0 0 0 0	114b	84.6k	35	Dec 26	17:17:59.623			
localhost	*0	*0	*0	*0	0	2 0	0.0%	1.1%	0	393G 39.0M 0 0 0 0	297b	83.6k	35	Dec 26	17:18:00.624			
localhost	*0	*0	*0	*0	0	3 0	0.0%	1.1%	0	393G 36.0M 0 0 0 0	323b	84.0k	35	Dec 26	17:18:01.619			
localhost	*0	*0	*0	*0	0	1 0	0.0%	1.1%	0	393G 36.0M 0 0 0 0	259b	83.2k	35	Dec 26	17:18:02.622			
localhost	*0	5	*0	*0	0	5 0	0.0%	1.1%	0	393G 44.0M 0 0 0 0	2.22k	88.4k	35	Dec 26	17:18:03.622			
localhost	*0	*0	*0	*0	0	2 0	0.0%	1.1%	0	393G 44.0M 0 0 0 0	246b	83.5k	35	Dec 26	17:18:04.620			
localhost	*0	*0	*0	*0	0	2 0	0.0%	1.1%	0	393G 42.0M 0 0 0 0	297b	83.4k	35	Dec 26	17:18:05.623			
localhost	*0	*0	*0	*0	0	0 0	0.0%	1.1%	0	393G 43.0M 0 0 0 0	111b	82.8k	35	Dec 26	17:18:06.625			
localhost	*0	*0	*0	*0	0	1 0	0.0%	1.1%	0	393G 43.0M 0 0 0 0	112b	83.3k	35	Dec 26	17:18:07.622			
localhost	*0	*0	*0	*0	0	2 0	0.0%	1.1%	0	393G 43.0M 0 0 0 0	307b	83.5k	35	Dec 26	17:18:08.624			
	host insert query update delete getmore command dirty used flushes vsize res qrw arw net_in net_out conn time																	
localhost	*0	*0	*0	*0	0	1 0	0.0%	1.1%	0	393G 41.0M 0 0 0 0	112b	83.6k	35	Dec 26	17:18:09.617			
localhost	*0	*0	*0	*0	0	2 0	0.0%	1.1%	0	393G 41.0M 0 0 0 0	296b	83.3k	35	Dec 26	17:18:10.622			
localhost	18045	*0	*0	*0	0	3 0	0.1%	1.3%	0	393G 49.0M 0 0 0 0	1.70m	85.5k	35	Dec 26	17:18:11.619			
localhost	3621	4	*0	*0	0	3 0	0.2%	1.3%	0	393G 51.0M 0 0 0 0	351k	88.3k	35	Dec 26	17:18:12.624			
localhost	*0	*0	*0	*0	0	4 0	0.2%	1.3%	0	393G 45.0M 0 0 0 0	303b	87.6k	35	Dec 26	17:18:13.620			
localhost	*0	*0	*0	*0	0	1 0	0.2%	1.3%	0	393G 45.0M 0 0 0 0	245b	83.2k	35	Dec 26	17:18:14.622			
localhost	*0	*0	*0	*0	0	3 0	0.2%	1.3%	0	393G 45.0M 0 0 0 0	298b	83.7k	35	Dec 26	17:18:15.622			
localhost	*0	*0	*0	*0	0	1 0	0.2%	1.3%	0	393G 45.0M 0 0 0 0	112b	83.1k	35	Dec 26	17:18:16.621			
localhost	*0	*0	*0	*0	0	0 0	0.2%	1.3%	0	393G 45.0M 0 0 0 0	111b	82.7k	35	Dec 26	17:18:17.625			
localhost	*0	*0	*0	*0	0	3 0	0.2%	1.3%	0	393G 45.0M 0 0 0 0	309b	84.1k	35	Dec 26	17:18:18.620			

Análisis del comando mongotop al importar “act-grupal-countries-small.json”

Al momento de importar los datos, se nos presenta un proceso en dos fases que reflejan el tiempo total empleado, el cual es la suma del tiempo invertido en las operaciones de lectura y escritura realizadas por el procesador en ese instante preciso.

Primera fase: En esta etapa, se utilizó la operación de lectura durante 60 milisegundos y en escritura de 0 milisegundos.

Segunda fase: En esta etapa, se utilizó la operación de lectura durante 13 milisegundos y una escritura de 2 milisegundos.

Análisis:

A partir de los datos obtenidos, podemos concluir que las operaciones cuando las colecciones son de menor complejidad o tamaño se pueden ejecutar en un mismo bloque, para este caso se uso mas tiempo para la lectura y poco segundos para la escritura.

```
mongotop 10
+-----+
| config.system.sessions | 0ms  0ms  0ms |
| config.transactions   | 0ms  0ms  0ms |
| countries.act-grupal-countries-big | 0ms  0ms  0ms |
| countries.act-grupal-countries-small | 0ms  0ms  0ms |
| inspections.act-grupal-city_inspections | 0ms  0ms  0ms |
| local.system.replset   | 0ms  0ms  0ms |
+-----+
          ns      total    read    write   2024-12-26T17:20:43-06:00
countries.act-grupal-countries-small
  config.collections    7ms   60ms   0ms
  admin.$cmd.aggregate  0ms   0ms   0ms
  admin.atlascli        0ms   0ms   0ms
  admin.system.version  0ms   0ms   0ms
  config.system.sessions 0ms   0ms   0ms
  config.transactions   0ms   0ms   0ms
countries.act-grupal-countries-big
  config.collections    0ms   0ms   0ms
  admin.$cmd.aggregate  0ms   0ms   0ms
  admin.atlascli        0ms   0ms   0ms
  admin.system.version  0ms   0ms   0ms
  config.system.sessions 0ms   0ms   0ms
  config.transactions   0ms   0ms   0ms
inspections.act-grupal-city_inspections
  config.collections    0ms   0ms   0ms
  admin.$cmd.aggregate  0ms   0ms   0ms
  admin.atlascli        0ms   0ms   0ms
  admin.system.version  0ms   0ms   0ms
  config.system.sessions 0ms   0ms   0ms
  config.transactions   0ms   0ms   0ms
local.system.replset
  config.collections    0ms   0ms   0ms
  admin.$cmd.aggregate  0ms   0ms   0ms
  admin.atlascli        0ms   0ms   0ms
  admin.system.version  0ms   0ms   0ms
  config.system.sessions 0ms   0ms   0ms
  config.transactions   0ms   0ms   0ms
+-----+
          ns      total    read    write   2024-12-26T17:20:53-06:00
countries.act-grupal-countries-small
  admin.$cmd.aggregate  0ms   16ms   2ms
  admin.atlascli        0ms   0ms   0ms
  admin.system.version  0ms   0ms   0ms
  config.collections    0ms   0ms   0ms
  config.system.sessions 0ms   0ms   0ms
  config.transactions   0ms   0ms   0ms
countries.act-grupal-countries-big
  config.collections    0ms   0ms   0ms
  admin.$cmd.aggregate  0ms   0ms   0ms
  admin.atlascli        0ms   0ms   0ms
  admin.system.version  0ms   0ms   0ms
  config.system.sessions 0ms   0ms   0ms
  config.transactions   0ms   0ms   0ms
inspections.act-grupal-city_inspections
  config.collections    0ms   0ms   0ms
  admin.$cmd.aggregate  0ms   0ms   0ms
  admin.atlascli        0ms   0ms   0ms
  admin.system.version  0ms   0ms   0ms
  config.system.sessions 0ms   0ms   0ms
  config.transactions   0ms   0ms   0ms
local.system.replset
  config.collections    0ms   0ms   0ms
  admin.$cmd.aggregate  0ms   0ms   0ms
  admin.atlascli        0ms   0ms   0ms
  admin.system.version  0ms   0ms   0ms
  config.system.sessions 0ms   0ms   0ms
  config.transactions   0ms   0ms   0ms
+-----+
          ns      total    read    write   2024-12-26T17:21:03-06:00
admin.$cmd.aggregate
  admin.atlascli        0ms   0ms   0ms
  admin.system.version  0ms   0ms   0ms
  config.collections    0ms   0ms   0ms
  config.system.sessions 0ms   0ms   0ms
  config.transactions   0ms   0ms   0ms
countries.act-grupal-countries-big
  config.collections    0ms   0ms   0ms
  admin.$cmd.aggregate  0ms   0ms   0ms
  admin.atlascli        0ms   0ms   0ms
  admin.system.version  0ms   0ms   0ms
  config.system.sessions 0ms   0ms   0ms
  config.transactions   0ms   0ms   0ms
countries.act-grupal-countries-small
  config.collections    0ms   0ms   0ms
  admin.$cmd.aggregate  0ms   0ms   0ms
  admin.atlascli        0ms   0ms   0ms
  admin.system.version  0ms   0ms   0ms
  config.system.sessions 0ms   0ms   0ms
  config.transactions   0ms   0ms   0ms
inspections.act-grupal-city_inspections
  config.collections    0ms   0ms   0ms
  admin.$cmd.aggregate  0ms   0ms   0ms
  admin.atlascli        0ms   0ms   0ms
  admin.system.version  0ms   0ms   0ms
  config.system.sessions 0ms   0ms   0ms
  config.transactions   0ms   0ms   0ms
local.system.replset
  config.collections    0ms   0ms   0ms
  admin.$cmd.aggregate  0ms   0ms   0ms
  admin.atlascli        0ms   0ms   0ms
  admin.system.version  0ms   0ms   0ms
  config.system.sessions 0ms   0ms   0ms
  config.transactions   0ms   0ms   0ms
```

Análisis del comando mongostat --discover al importar “act-grupal-countries-small.json”

La herramienta reportó un promedio de [número] operaciones por segundo durante la importación de la colección. Se registró una inserción inicial y 5 consultas adicionales para validar los datos. La instancia local de MongoDB manejó la carga sin problemas, utilizando solo 1.3% de memoria y un promedio de 0.0% de CPU. El análisis del tráfico de red mostró 3.84 KB de datos entrantes y 91.3 KB de datos salientes.

mongostat --discover																	
host	insert	query	update	delete	getmore	command	dirty	used	flushes	vsize	res	qrw	arw	net_in	net_out	conn	time
localhost	*0	*0	*0	*0	0	4 0	0.0%	1.3%	0	393G 41.0M 0 0 0 0	457b	83.2K	35 Dec 26 17:20:21.020				
localhost	*0	*0	*0	*0	0	1 0	0.0%	1.3%	0	393G 42.0M 0 0 0 0	112b	83.2K	35 Dec 26 17:20:22.618				
localhost	*0	*0	*0	*0	0	1 0	0.0%	1.3%	0	393G 38.0M 0 0 0 0	112b	83.3K	35 Dec 26 17:20:24.617				
localhost	no data received																
localhost	*0	*0	*0	*0	0	1 0	0.0%	1.3%	0	393G 38.0M 0 0 0 0	244b	82.7K	35 Dec 26 17:20:25.625				
localhost	*0	*0	*0	*0	0	3 0	0.0%	1.3%	0	393G 38.0M 0 0 0 0	298b	83.7K	35 Dec 26 17:20:26.624				
localhost	*0	*0	*0	*0	0	1 0	0.0%	1.3%	0	393G 38.0M 0 0 0 0	112b	83.3K	35 Dec 26 17:20:27.621				
localhost	*0	*0	*0	*0	0	2 0	0.0%	1.3%	0	393G 38.0M 0 0 0 0	174b	83.4K	35 Dec 26 17:20:28.620				
localhost	host insert query update delete getmore command dirty used flushes vsize res qrw arw net_in net_out conn time																
localhost	*0	*0	*0	*0	0	1 0	0.0%	1.3%	0	393G 38.0M 0 0 0 0	245b	83.3K	35 Dec 26 17:20:29.621				
localhost	*0	*0	*0	*0	0	2 0	0.0%	1.3%	0	393G 38.0M 0 0 0 0	164b	83.4K	35 Dec 26 17:20:30.620				
localhost	*0	*0	*0	*0	0	2 0	0.0%	1.3%	0	393G 39.0M 0 0 0 0	307b	83.5K	35 Dec 26 17:20:31.622				
localhost	*0	*0	*0	*0	0	2 0	0.0%	1.3%	0	393G 39.0M 0 0 0 0	260b	83.6K	35 Dec 26 17:20:32.620				
localhost	*0	*0	*0	*0	0	4 0	0.0%	1.3%	0	393G 39.0M 0 0 0 0	447b	87.0K	35 Dec 26 17:20:33.625				
localhost	*0	*0	*0	*0	0	1 0	0.0%	1.3%	0	393G 39.0M 0 0 0 0	112b	83.2K	35 Dec 26 17:20:34.624				
localhost	*0	*0	*0	*0	0	2 0	0.0%	1.3%	0	393G 39.0M 0 0 0 0	247b	83.7K	35 Dec 26 17:20:35.619				
localhost	*0	5	*0	*0	0	4 0	0.0%	1.3%	0	393G 42.0M 0 0 0 0	2.22k	84.4K	35 Dec 26 17:20:36.626				
localhost	*0	*0	*0	*0	0	1 0	0.0%	1.3%	0	393G 41.0M 0 0 0 0	112b	83.3K	35 Dec 26 17:20:37.623				
localhost	*0	*0	*0	*0	0	2 0	0.0%	1.3%	0	393G 41.0M 0 0 0 0	174b	83.4K	35 Dec 26 17:20:38.622				
localhost	host insert query update delete getmore command dirty used flushes vsize res qrw arw net_in net_out conn time																
localhost	*0	*0	*0	*0	0	1 0	0.0%	1.3%	0	393G 41.0M 0 0 0 0	245b	83.1K	35 Dec 26 17:20:39.625				
localhost	*0	*0	*0	*0	0	2 0	0.0%	1.3%	0	393G 41.0M 0 0 0 0	164b	83.6K	35 Dec 26 17:20:40.622				
localhost	*0	*0	*0	*0	0	2 0	0.0%	1.3%	0	393G 41.0M 0 0 0 0	306b	83.4K	35 Dec 26 17:20:41.625				
localhost	*0	*0	*0	*0	0	2 0	0.0%	1.3%	0	393G 41.0M 0 0 0 0	260b	83.4K	35 Dec 26 17:20:42.625				
localhost	1	5	*0	*0	0	7 0	0.0%	1.3%	0	393G 43.0M 0 0 0 0	3.84k	91.3K	35 Dec 26 17:20:43.623				
localhost	*0	*0	*0	*0	0	1 0	0.0%	1.3%	0	393G 42.0M 0 0 0 0	112b	83.1K	35 Dec 26 17:20:44.622				
localhost	*0	*0	*0	*0	0	1 0	0.0%	1.3%	0	393G 42.0M 0 0 0 0	244b	83.0K	35 Dec 26 17:20:45.626				
localhost	*0	*0	*0	*0	0	3 0	0.0%	1.3%	0	393G 42.0M 0 0 0 0	298b	83.9K	35 Dec 26 17:20:46.623				
localhost	*0	*0	*0	*0	0	0 0	0.0%	1.3%	0	393G 43.0M 0 0 0 0	111b	82.8K	35 Dec 26 17:20:47.625				
localhost	*0	*0	*0	*0	0	2 0	0.0%	1.3%	0	393G 43.0M 0 0 0 0	174b	83.5K	35 Dec 26 17:20:48.624				
localhost	host insert query update delete getmore command dirty used flushes vsize res qrw arw net_in net_out conn time																

Exportar la base de datos

Usando comando mongoexport

Para validar el funcionamiento de este comando, se siguió la sugerencia de la actividad y se generó el siguiente comando: “mongoexport --db inspections --collection act-grupal-city_inspections --out inspections.json;” Al obtener el resultado esperado, se adjunta una captura de pantalla para mostrar la operación.

```
apple ~ % /opt/homebrew/var/log/mongodb on master
> mongoexport --db inspections --collection act-grupal-city_inspections --out inspections.json;
2024-12-26T18:30:13.913-0600    connected to: mongodb://localhost/
2024-12-26T18:30:14.511-0600    exported 81047 records
>
apple ~ % /opt/homebrew/var/log/mongodb on master
> ls -l
total 47928
-rw-r--r--  1 leonard  admin  24269552 Dec 26 18:30 inspections.json
-rw-----  1 root     admin    246908 Dec 26 18:30 mongo.log
-rw-r--r--  1 leonard  admin        0 Dec 16 22:15 output.log
>
```

Restaurar una base de datos

Usando comando mongoexport

Siguiendo las indicaciones de la actividad, se descomprimió el archivo comprimido 'act-grupal-people.zip', obteniendo el archivo en formato BSON 'people.bson'. A continuación, se ejecutó el comando 'mongorestore people.bson' desde la ruta donde se encontraba el archivo descomprimido para verificar su correcto funcionamiento.

```
mongorestore people.json  
2024-12-26T18:48:44.098-0600 checking for collection data in people.json  
2024-12-26T18:48:44.099-0600 reading metadata for test.people from people.metadata.json  
2024-12-26T18:48:44.145-0600 Restoring test.people from people.json  
2024-12-26T18:48:46.504-0600 finished restoring test.people (1000000 documents, 0 failures)  
2024-12-26T18:48:46.504-0600 no indexes to restore for collection test.people  
2024-12-26T18:48:46.504-0600 1000000 document(s) restored successfully. 0 document(s) failed to restore.
```

The screenshot shows the MongoDB Compass interface. The left sidebar displays connections, with 'cluster0.nhica.mongodb.net' and 'LOCAL' selected. Under 'LOCAL', several databases are listed: admin, config, countries, inspections, local, and test. The 'people' collection is currently selected under the test database. The main pane shows the 'Documents' tab with 1.0M documents. A search bar at the top says 'Type a query: { field: 'value' } or Generate query'. Below it are buttons for ADD DATA, EXPORT DATA, UPDATE, and DELETE. The results list shows five documents, each with a unique _id, timestamped padding, signups, points, and names ('Dan', 'Nick', 'Nick', 'Wendy', and another entry with _id '57605d5dc3d5a2429db0bcff'). Each document has edit and delete icons to its right.

Caso de uso

Restricción de terrazas en Madrid por covid-19. Contratan a vuestro equipo para actualizar las restricciones de ciertos locales y terrazas en Madrid por cuestiones del covid-19. Los datos para actualizar están en el fichero llamado act-grupal-openDataLocalesMadrid.csv y se os pide que consolidéis dichos cambios en una base de datos MongoDB llamada Madrid con la colección Terrazas. La información del dataset a utilizar lo tenéis en la siguiente URL: Open Data Censo de locales, sus actividades y terrazas de hostelería y restauración (Terrazas).

Se genera la Base de datos Madrid y su colección Terrazas.

```
mongosh mongodb://127.0.0.1/ + ▾
Madrid> db.getCollectionNames()
[]
Madrid> db.createCollection("Terrazas")
{ ok: 1 }
Madrid> db.getCollectionNames()
[ 'Terrazas' ]
Madrid> |
```

```
mongosh mongodb://127.0.0.1/ + ▾ ×
Madrid> load("D:\\mongo-imports\\act-grupal-openDataLocalesMadrid.js")
true
Madrid> datos_insertar[0]
{
  _id: 7,
  id_local: 280067128,
  id_distrito_local: 20,
  desc_distrito_local: 'SAN BLAS-CANILLEJAS',
  desc_barrio_local: 'ROSAS',
  clase_vial_edificio: 'CALLE',
  num_edificio: 177,
  Cod_Postal: 28022,
  coordenada_x_local: '448900,55',
  coordenada_y_local: '4474755,41',
  desc_situacion_local: 'Abierto',
  Nombre: 'SR',
  id_período_terraza: 2,
  desc_período_terraza: 'Estacional',
  id_situacion_terraza: 1,
  desc_situacion_terraza: 'Abierta',
  Superficie_ES: '7,2',
  DESC_CLASE: 'CALLE',
  DESC_NOMBRE: 'SOFIA',
  num_terraza: 177,
  cal_terraza: 'A',
  desc ubicación_terraza: 'Acera',
  hora_ini_LJ_es: '8:00:00',
  hora_fin_LJ_es: '1:30:00',
  hora_ini_LJ_ra: '',
  hora_fin_LJ_ra: '',
  hora_ini_VS_es: '8:00:00',
  hora_fin_VS_es: '1:30:00',
  hora_ini_VS_ra: '',
  hora_fin_VS_ra: '',
  mesas_aux_es: 0,
  mesas_aux_ra: '',
  mesas_es: 5,
  mesas_ra: '',
  sillas_es: 10,
  sillas_ra: 'A'
}
Madrid>
```

```
mongosh mongoDB://127.0.0.1 + v
Madrid> db.Terrazas.insertMany(datos_insertar)
{
  acknowledged: true,
  insertedIds: [
    "0": 7,
    "1": 31,
    "2": 33,
    "3": 30,
    "4": 36,
    "5": 48,
    "6": 41,
    "7": 44,
    "8": 59,
    "9": 81,
    "10": 52,
    "11": 54,
    "12": 56,
    "13": 58,
    "14": 59,
    "15": 68,
    "16": 63,
    "17": 66,
    "18": 67,
    "19": 68,
    "20": 69,
    "21": 72,
    "22": 73,
    "23": 74,
    "24": 76,
    "25": 77,
    "26": 78,
    "27": 79,
    "28": 88,
    "29": 81,
    "30": 82,
    "31": 83,
    "32": 84,
    "33": 85,
    "34": 86,
    "35": 87,
    "36": 88,
    "37": 89,
    "38": 91,
    "39": 95,
    "40": 98,
    "41": 101,
    "42": 102,
    "43": 103,
    "44": 104,
    "45": 105
  ]
}
```

Ejercicio #1:

Descripción: Esta consulta buscará en la colección "Terrazas" todos los documentos que representen locales en el barrio de "GUINDALERA" y que actualmente no estén "Cerrados". A todos estos documentos, les cambiará el estado del local a "Cerrado" y además agregará un nuevo campo indicando que la terraza también está "Cerrada". Como podemos demostrar en el código nos apoyamos con el método updateMany que nos permite editar todo el documento seleccionado, luego con el método \$set podemos establecer un nuevo campo llamado desc_situacion_local y otro llamado desc_situacion_terraza estableciendo de una vez que valor tendrá cumpliendo las condiciones.

```
db.Terrazas.updateMany(  
  {  
    "desc_barrio_local": "GUINDALERA",  
    "desc_situacion_local": { $ne: "Cerrado" }  
  },  
  {  
    $set: {  
      "desc_situacion_local": "Cerrado",  
      "desc_situacion_terraza": "Cerrada"  
    }  
  }  
);
```

```
Madrid> db.Terrazas.updateMany( { "desc_barrio_local": "GUINDALERA", "desc_situacion_local": { $ne: "Cerrado" } }, { $set: { "desc_situacion_local": "Cerrado", "desc_situacion_terraza": "Cerrada" } } );  
{  
  acknowledged: true,  
  insertedId: null,  
  matchedCount: 32,  
  modifiedCount: 32,  
  upsertedCount: 0  
}  
Madrid> |
```

Ejercicio #2:

Descripción: Esta consulta buscará todas las terrazas ubicadas en la acera y, para cada una de ellas, agregará (o actualizará) un campo llamado "inspeccionar". El valor de este campo será true si la terraza tiene más de 10 mesas, y false en caso contrario. Como podemos demostrar en el código nos apoyamos con el método updateMany que nos permite editar todo el documento seleccionado, luego con el método \$set podemos establecer un nuevo campo llamado inspeccionar y con el método \$cond podemos realizar un condicional para y definir que valor tendrá el nuevo campo.

```
db.Terrazas.updateMany(  
  { "desc_ubicacion_terraza": "Acera" }  
,  
  [  
    {  
      $set:{  
        "inspeccionar":{  
          $cond: {  
            if: { $gt: ["$mesas_es", 10] },  
            then: true,  
            else: false  
          }  
        }  
      }  
    }  
  ]  
)
```

```
Madrid> db.Terrazas.updateMany(  
...   { "desc_ubicacion_terrazas": "Acera" }, // Filtrar por ubicación "Acera"  
...   [  
...     {  
...       $set: {  
...         "inspeccionar": {  
...           $cond: {  
...             if: { $gt: ["$mesas_es", 10] }, // Si mesas_es es mayor que 10  
...             then: true,  
...             else: false  
...           }  
...         }  
...       }  
...     }  
...   ]  
... );  
{  
  acknowledged: true,  
  insertedId: null,  
  matchedCount: 2319,  
  modifiedCount: 2319,  
  upsertedCount: 0  
}
```



Ejercicio #3:

Ejercicio #4:

<p>Descripción: La consulta db.Terrazas.updateMany() busca actualizar múltiples registros en la colección "Terrazas" donde el campo "inspeccionar" tenga el valor "false". Utilizando el operador \$set, se agrega un nuevo campo llamado "estado" cuyo valor se determina mediante una estructura condicional (\$cond) anidada. Esta estructura evalúa el número de sillas en cada terraza y asigna un valor de 1, 2 o 3 al campo "estado" según si el número de sillas es menor a 10, entre 10 y 20, o mayor a 20 respectivamente, como podemos mencionar nos apoyamos con el método updateMany para poder actualizar todo el documento con los nuevos valores para este caso estado.</p>	<pre>db.Terrazas.updateMany({ "inspeccionar": false }, [{ \$set: { "estado": { "\$cond": { "if": { \$lt: ["\$sillas_es", 10] }, "then": 1, "else": { "\$cond": { "if": { \$and: [{ \$gte: ["\$sillas_es", 10], { \$lte: ["\$sillas_es", 20] }] } }, "then": 2, "else": 3 } } } } }]))</pre>	<pre>db.Terrazas.updateMany({ "inspeccionar": false }, [{ \$set: { "estado": { "\$cond": { "if": { \$lt: ["\$sillas_es", 10] }, "then": 1, "else": { "\$cond": { "if": { \$and: [{ \$gte: ["\$sillas_es", 10], { \$lte: ["\$sillas_es", 20] }] } }, "then": 2, "else": 3 } } } } }] }) { acknowledged: true, insertedId: null, matchCount: 1500, modifiedCount: 1500, upsertedCount: 0 }</pre>
--	---	--

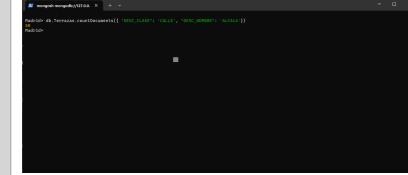
Ejercicio #5:

<p>Descripción: La consulta permite realizar un proceso de dos etapas. Primero, se selecciona la colección Terrazas como el objetivo de la operación. Luego, se utiliza la función aggregate para iniciar una tubería de procesamiento. Dentro de esta tubería, el operador \$set se encarga de setear el campo denominado hora_fin_LJ_es a cada documento de la colección. Este campo se inicializa con el valor constante '00:00:00', lo que equivale a establecer una hora de finalización uniforme para todos los registros.</p>	<pre>db.Terrazas.aggregate([{ \$set: { hora_fin_LJ_es: '00:00:00' } }]);</pre>	<pre>>_MONGOSH > db.Terrazas.aggregate([{ \$set: { hora_fin_LJ_es: '00:00:00' } }]); < { _id: 7, id_local: 280067128, id_distrito_local: 20, desc_distrito_local: 'SAN BLAS-CANILLEJAS', desc_barrio_local: 'ROSAS', clase_vial_edificio: 'CALLE', num_edificio: 177, Cod.Postal: 28022, coordenada_x_local: '448900,55', coordenada_y_local: '4474753,41', desc_situacion_local: 'Abierto', Nombre: 'SR', id_periodo_terraza: 2, desc_periodo_terraza: 'Estacional', id_situacion_terraza: 1, desc_situacion_terraza: 'Abierta', Superficie_ES: '7,2', DESC_CLASE: 'CALLE', DESC_NOMBRE: 'SOFIA',</pre>
---	--	---

Ejercicio #6:

<p>Descripción: La consulta opera sobre la colección "Terrazas", especificada mediante db.Terrazas. El método updateMany() se encarga de actualizar múltiples documentos que cumplen con un criterio de búsqueda, a diferencia de updateOne() que solo actualiza el primer documento encontrado. El criterio de búsqueda, definido en el primer argumento de updateMany(), busca todos los documentos dentro de la colección "Terrazas" donde el campo "hora_fin_VS_es" sea igual a la cadena '2:30:00'.</p>	<pre>db.Terrazas.updateMany({ "hora_fin_VS_es": '2:30:00' }, [{ \$set: { hora_fin_VS_es: '2:00:00' } }]);</pre>	<pre>>_MONGOSH > db.Terrazas.updateMany({ "hora_fin_VS_es": '2:30:00' }, [{ \$set: { hora_fin_VS_es: '2:00:00' } }]); < { acknowledged: true, insertedId: null, matchedCount: 1550, modifiedCount: 1550, upsertedCount: 0 }</pre>
---	---	--

Ejercicio #7:

<p>Descripción: La función db.Terrazas.updateMany() en MongoDB se encarga de actualizar múltiples documentos dentro de la colección "Terrazas" en la base de datos actual. El método updateMany() requiere dos argumentos principales: un filtro de búsqueda y una operación de actualización. El primer argumento, { \$and: [{ DESC_CLASE: 'CALLE' }, { DESC_NOMBRE: 'ALCALA' }] }, define el criterio de búsqueda. Utiliza el operador \$and para asegurar que ambas condiciones se cumplan: que el campo DESC_CLASE sea igual a 'CALLE' y que el campo DESC_NOMBRE sea igual a 'ALCALA'. Esto selecciona los documentos que cumplen con ambos criterios simultáneamente. El segundo argumento, [{ \$set: { inspeccionar: true } }], especifica la operación de actualización. El operador \$set se utiliza para crear un nuevo campo llamado inspeccionar (si no existe) o modificar su valor a true en los documentos que coinciden con el filtro. En pocas palabras, la consulta actualiza todos los documentos en la colección "Terrazas" donde DESC_CLASE es 'CALLE' y DESC_NOMBRE es 'ALCALA', añadiendo o modificando el campo inspeccionar a true</p>	<pre>db.Terrazas.updateMany({ \$and: [{ DESC_CLASE: 'CALLE' }, { DESC_NOMBRE: 'ALCALA' }] }, [{ \$set: { inspeccionar: true } }]);</pre>	<pre>> db.Terrazas.updateMany({ \$and: [{ DESC_CLASE: 'CALLE' }, { DESC_NOMBRE: 'ALCALA' }] }, [{ \$set: { inspeccionar: true } }]); < { acknowledged: true, insertedId: null, matchedCount: 50, modifiedCount: 39, upsertedCount: 0 } Madrid></pre> 
--	--	--

Ejercicio #8:

<p>Descripción: La función db.Terrazas.updateMany() se utiliza para actualizar múltiples documentos dentro de la colección "Terrazas".</p> <p>Especificamente, actualiza aquellos documentos donde el campo desc_situacion_terraza sea igual a "Abierta", lo que asegura que solo se modifiquen los locales con terrazas abiertas.</p> <p>La operación consiste en añadir, mediante el operador \$set, un nuevo campo llamado revision con el valor { prox_inspeccion: 10, puntuacion: 80, comentario: "separar las mesas" }, que contiene información sobre la revisión, incluyendo la próxima inspección, la puntuación y un comentario.</p>	<pre>db.Terrazas.updateMany({ "desc_situacion_terraza": "Abierta" }, { \$set: { revision: { prox_inspeccion: 10, puntuacion: 80, comentario: "separar las mesas" } } });</pre>	<pre>> db.Terrazas.updateMany({ "desc_situacion_terraza": "Abierta" }, { \$set: { revision: { prox_inspeccion: 10, puntuacion: 80, comentario: "separar las mesas" } } }); < { acknowledged: true, insertedId: null, matchedCount: 2953, modifiedCount: 2953, upsertedCount: 0 }</pre>
---	--	--

Ejercicio #9:

Descripción:	<p>La consulta db.Terrazas.aggregate() es la sintaxis básica para ejecutar una consulta de agregación en la colección "Terrazas", donde aggregate() toma un array de etapas que se ejecutan secuencialmente para transformar los datos. La etapa \$match: { desc_distrito_local: "VILLAVERDE" } actúa como una cláusula WHERE en SQL, filtrando los documentos de "Terrazas" donde el campo desc_distrito_local es igual a "VILLAVERDE", seleccionando así todas las terrazas en el distrito de Villaverde. Finalmente, la etapa \$merge: { into: "Zona1", on: "_id", whenMatched: "replace", whenNotMatched: "insert" } combina los resultados con la colección "Zona1"; into: "Zona1" especifica la colección destino; on: "_id" define el campo de coincidencia, buscando documentos en "Zona1" con el mismo _id que los filtrados de "Terrazas"; whenMatched: "replace" indica que el documento existente en "Zona1" se reemplazará por el de "Terrazas" si hay coincidencia; y whenNotMatched: "insert" indica que el documento de "Terrazas" se insertará en "Zona1" si no hay coincidencia</p>	
Código	<pre>db.Terrazas.aggregate([{ \$match: { desc_distrito_local: "VILLAVERDE" } }, { \$merge: { into: "Zona1", on: "_id", whenMatched: "replace", whenNotMatched: "insert" } }]) // Validar db["Zona1"].find().count()</pre>	
Evidencias	<pre>> db.Terrazas.aggregate([{ \$match: { desc_distrito_local: "VILLAVERDE" } }, { \$merge: { into: "Zona1", on: "_id", whenMatched: "replace", whenNotMatched: "insert" } }]) < Madrid></pre>	
	<pre>> db["Zona1"].find().count() < 119 Madrid ></pre>	
	<pre>> db["Zona1"].find() < _id: 54, id_local: 280062306, id_distrito_local: 17, desc_distrito_local: "VILLAVERDE", desc_barrio_local: "SAN CRISTOBAL", clase_via_edificio: "CALLE", num_edificio: 119, Cod_Postal: 28021, coordenada_x_local: "441294,52", coordenada_y_local: "4465802,5", desc_situacion_local: "Abierto", Nombre: "BAR LAS TORRES", id_período_terraza: 2, desc_período_terraza: "Estacional", id_situacion_terraza: 1, desc_situacion_terraza: "Abierta", Superficie_ES: "21,66", DESC_CLASE: "CALLE", DESC_NOMBRE: "MONCADA", num_terraza: 119, cal_terraza: "", desc ubicacion_terraza: "Acera", hora_ini_L3_es: "01:00:00", hora_fin_L3_es: "01:00:00",</pre>	

Ejercicio #10:

Descripción:	<p>La línea db.Terrazas.aggregate, es una agregación, la etapa \$match: { ... } filtra los documentos de "Terrazas", permitiendo pasar a la siguiente etapa solo aquellos que cumplen con las condiciones especificadas: desc_distrito_local: "SALAMANCA" (el campo desc_distrito_local debe ser igual a "SALAMANCA") y desc_barrio_local: "CASTELLANA" (el campo desc_barrio_local debe ser igual a "CASTELLANA"), recordando que MongoDB distingue entre mayúsculas y minúsculas, por lo que los valores deben coincidir exactamente con los de la base de datos.</p> <p>Finalmente, la etapa { \$out: "Zona2" } escribe los documentos resultantes del \$match en una nueva colección llamada "Zona2"; si esta colección no existe, se crea automáticamente, y si ya existe, se sobrescribe completamente con los nuevos resultados.</p>	
Código	<pre>db.Terrazas.aggregate([{ \$match: { desc_distrito_local: { \$regex: "salamanca", \$options: "i" }, // "i" para insensible a mayúsculas y minúsculas desc_barrio_local: { \$regex: "castellana", \$options: "i" } }, { \$merge: { into: "Zona2", // Nombre de la colección destino on: "_id", // Campo para la coincidencia (si se omite, se insertan nuevos documentos) whenMatched: "replace", // "replace", "keepExisting", "fail", "merge" (acción cuando hay coincidencia) whenNotMatched: "insert" // "insert", "discard", "fail" (acción cuando no hay coincidencia) } } }) // Consulta para validar db["Zona2"].find().count()</pre>	

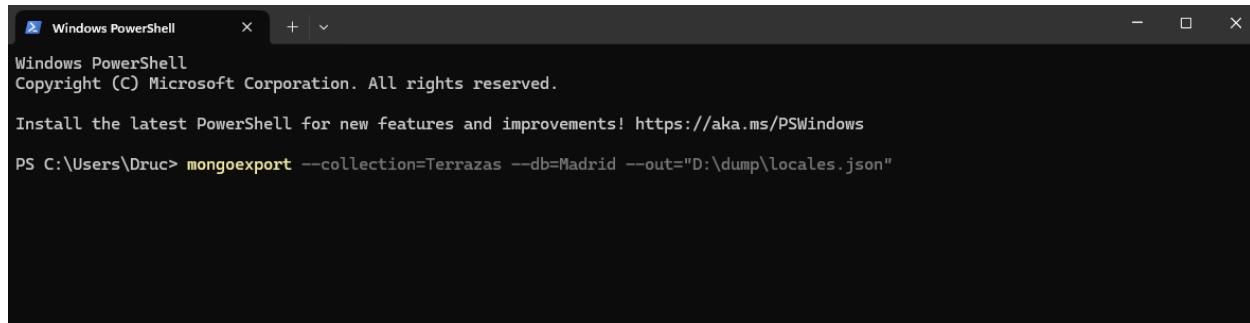
Evidencias	<pre>> db.Terrazas.aggregate([{ \$match: { desc_distrito_local: { \$regex: "salamanca", \$options: "i" }, // "i" para insensible a mayúsculas y minúsculas desc_barrio_local: { \$regex: "castellana", \$options: "i" } } }, { \$merge: { into: "Zona2", // Nombre de la colección destino on: "_id", // Campo para la coincidencia (si se omite, se insertan nuevos documentos) whenMatched: "replace", // "replace", "keepExisting", "fail", "merge" (acción cuando hay coincidencia) whenNotMatched: "insert" // "insert", "discard", "fail" (acción cuando no hay coincidencia) } }]) < Madrid</pre>	
	<pre>db["Zona2"].find().count() < 44 Madrid ></pre>	

Grafos

Explicación paso a paso

Para extraer la información desde mongo se realizó la ejecución del comando:

```
mongoexport --collection=Terrazas --db=Madrid --out="D:\dump\locales.json"
```



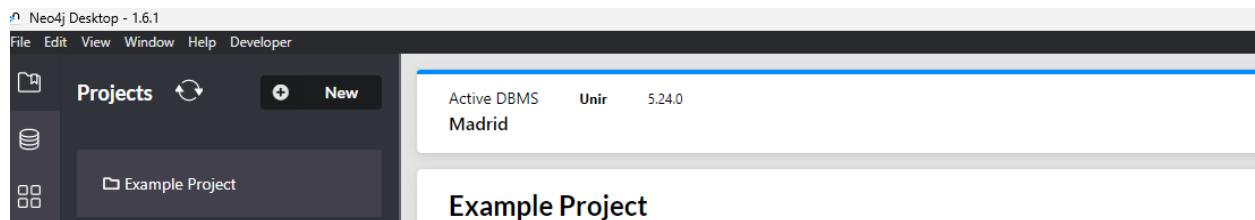
The screenshot shows a Windows PowerShell window with the following output:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Druc> mongoexport --collection=Terrazas --db=Madrid --out="D:\dump\locales.json"
```

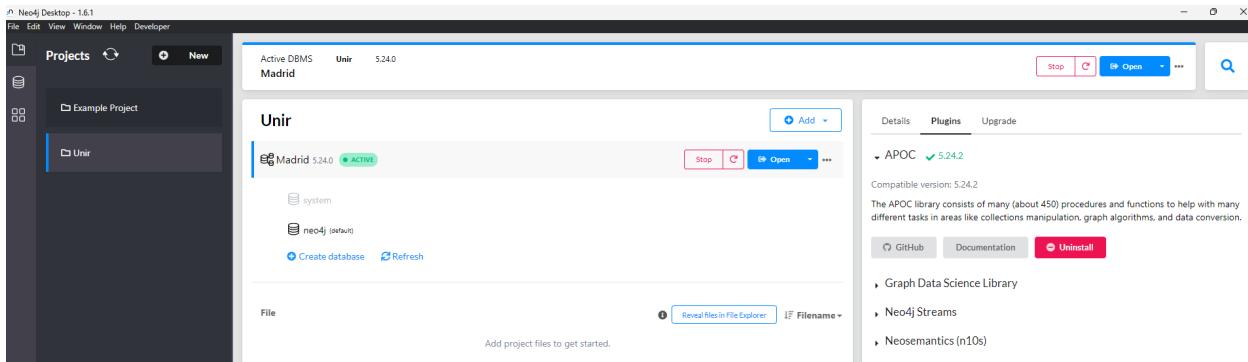
Para comenzar a trabajar en Neo4j primero tenemos que crear un proyecto



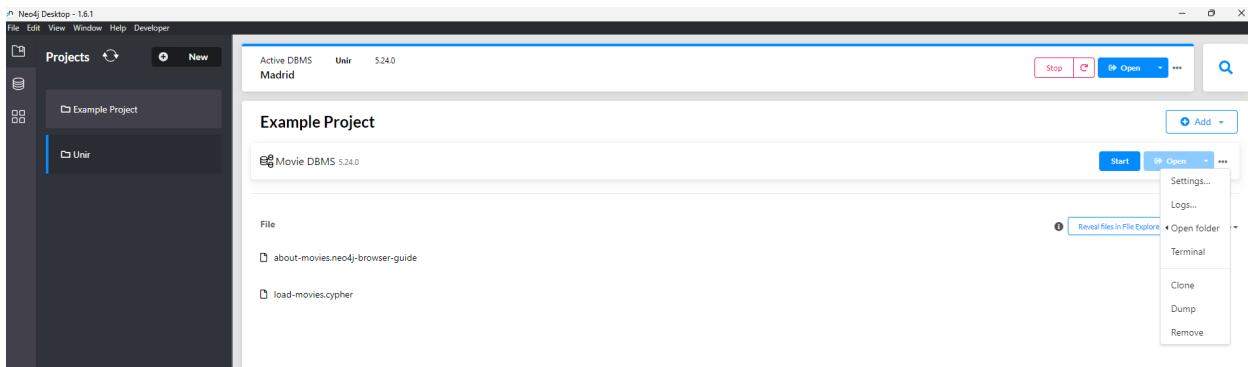
Para este caso creamos un proyecto con nombre unir y dentro creamos la base de datos, Madrid en el menú “Add”

Para poder trabajar con la información exportada desde mongo, agregamos el complemento “apoc” el cual nos permitirá trabajar con el archivo json resultante de la exportación

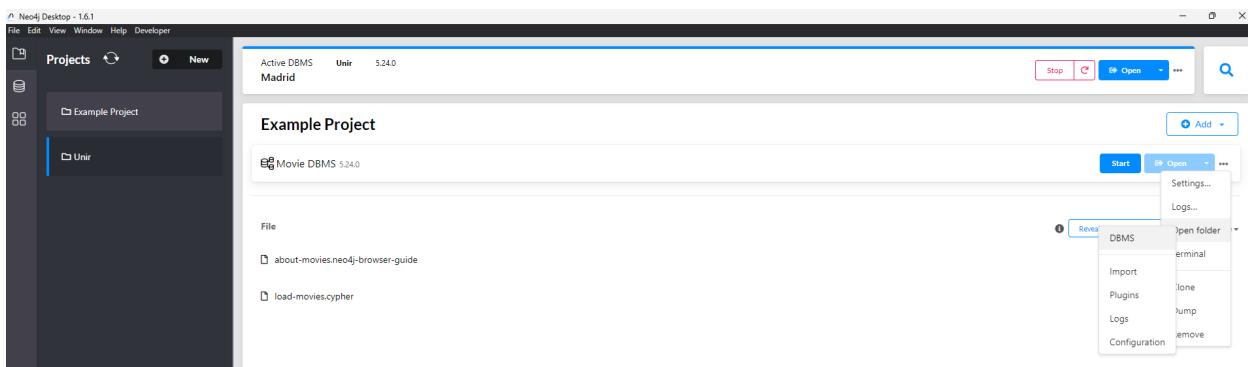
Detail	Value
Version	5.24.0
Edition	enterprise
Status	Active
IP address	localhost
Bolt port	7687
HTTP port	7474
HTTPS port	7473



La instalación nos pedirá reiniciar la base de datos, como ultimo paso, para hacer uso del archivo json, seleccionamos el menú que aparece al posicionarnos sobre el nombre de la base de datos



Y seleccionamos open **folder**, el cual nos mostrara diferentes carpetas



En esta seleccionamos Import y en ella pondremos el archivo json o csv resultante dependiendo el formato que se desee trabajar.

Una vez realizado esto, damos clic al botón “start” el cual nos enviar a la consola de **Neo4j Browser**, en el cual podremos hacer uso de **cypher** y las consultas **CQL**.

Modelo

Para representar la información en Neo4j y visualizar los locales de cada barrio y los tipos de terrazas que tienen, se puede modelar un grafo con los siguientes nodos y relaciones

Nodos:

1. Local: Representa un local de negocio.

Atributos:

- id_local (ID único del local)
- Nombre (Nombre del local)
- Cod_Postal (Código postal del local)
- desc_situacion_local (Estado del local: abierto, cerrado, etc.)
- clase_vial_edificio (Tipo de vial: Calle, etc.)
- coordenada_x_local (Coordenada X)
- coordenada_y_local (Coordenada Y)
- num_edificio (Número del edificio)
- desc_barrio_local (Barrio donde está ubicado el local)

2. Barrio: Representa el barrio donde se encuentra el local.

Atributos:

- desc_barrio_local (Nombre del barrio)
- id_distrito_local (ID del distrito)
- desc_distrito_local (Nombre del distrito)

3. Terraza: Representa una terraza asociada a un local.

Atributos:

- num_terraza (Número de la terraza)
- desc_ubicacion_terraza (Ubicación de la terraza)
- desc_situacion_terraza (Situación de la terraza: abierta, cerrada, etc.)
- desc_periodo_terraza (Periodo de la terraza: estacional, permanente, etc.)

- hora_ini_VS_es, hora_fin_VS_es, hora_ini_LJ_es, hora_fin_LJ_es, hora_ini_VS_ra, hora_fin_VS_ra, hora_ini_LJ_ra, hora_fin_LJ_ra (Horarios asociados a las terrazas)
- mesas_es, sillas_es, mesas_ra, sillas_ra (Cantidad de mesas y sillas)

Relaciones:

- ESTÁ_EN: Relaciona a un local con un barrio.

Atributos:

- id_local (ID del local)
- desc_barrio_local (Nombre del barrio)

- TIENE_TERRAZA: Relaciona un local con una terraza.

Atributos:

- num_terraza (Número de la terraza)
- desc_ubicacion_terraza (Ubicación de la terraza)
- estado (Estado de la terraza: abierta, cerrada, etc.)

Import y Visualización del Grafo

La consulta anterior creará un grafo con nodos Barrio, Local, Terraza y Revisión, y las relaciones **ESTÁ_EN**, **TIENE_TERRAZA**. Puedes visualizar el grafo generado en **Neo4j** usando su interfaz de usuario, y ver cómo se conectan los diferentes elementos. Si tienes una gran cantidad de datos, asegúrate de realizar las consultas de forma eficiente, usando WITH para manejar los datos por partes y evitar problemas de memoria.

Consulta CQL

```
CALL apoc.load.jsonArray('file:///locales.json') YIELD value AS locale
```

```
MERGE (b: Barrio {id_distrito_local: locale.id_distrito_local, desc_barrio_local: locale.desc_barrio_local, desc_distrito_local: locale.desc_distrito_local})
```

```
MERGE (l: Local {id_local: locale.id_local, Nombre: locale.Nombre, Cod_Postal: locale.Cod_Postal, desc_situacion_local: locale.desc_situacion_local, clase_vial_edificio:
```

```

locale.clase_vial_edificio, coordenada_x_local: locale.coordenada_x_local, coordenada_y_local:
locale.coordenada_y_local, num_edificio: locale.num_edificio, desc_barrio_local:
locale.desc_barrio_local})

MERGE (l)-[:ESTÁ_EN]->(b)

WITH l,b, locale

MERGE (t: Terraza {num_terraza: locale.num_terraza, desc_ubicacion_terraza:
locale.desc_ubicacion_terraza, desc_situacion_terraza: locale.desc_situacion_terraza,
desc_periodo_terraza: locale.desc_periodo_terraza})

MERGE (l)-[:TIENE_TERRAZA]->(t)

RETURN b, l, t LIMIT 100

```

Explicación detallada de cada parte del query:

- **Carga de datos con apoc.load.jsonArray:**

Se utiliza la función APOC para cargar un archivo JSON desde el sistema de archivos local, lo cual da como resultado una lista de objetos (representados por value).

- **Creación de los nodos:**

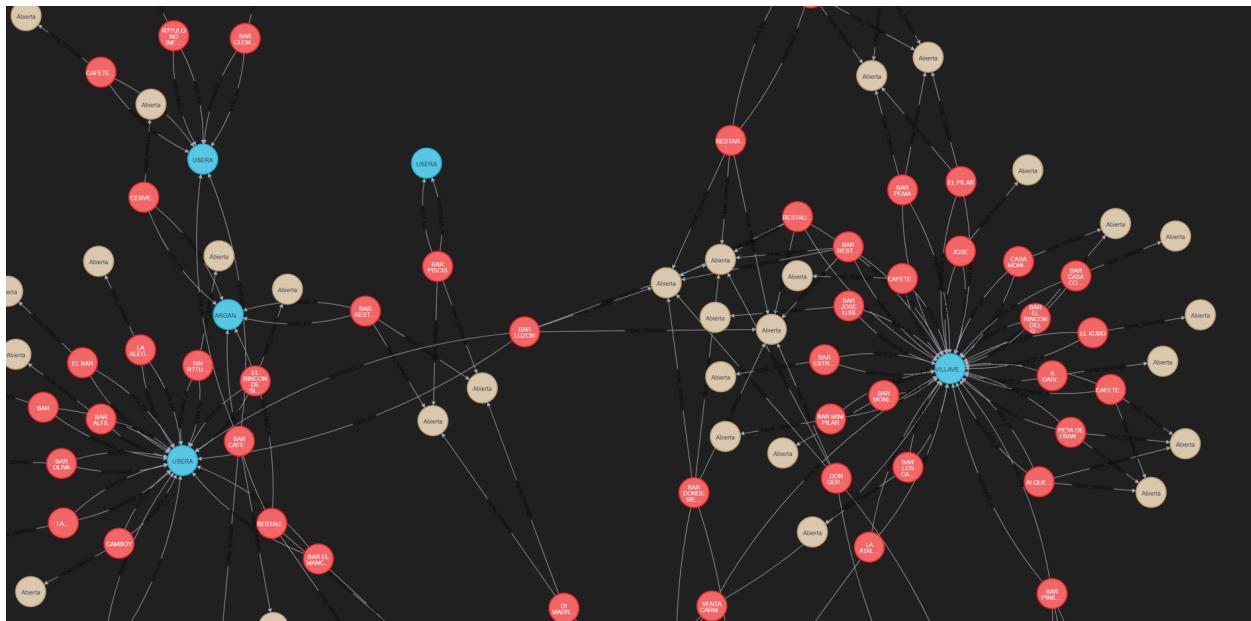
- Barrio: Usamos MERGE para crear un nodo de tipo Barrio que representa una zona geográfica (por ejemplo, un barrio o distrito). La combinación de id_distrito_local y desc_barrio_local se utiliza como clave única para evitar duplicados.
- Local: El nodo Local representa un establecimiento o negocio, identifiable por el id_local. También se incluyen otros atributos como el nombre del local, código postal, situación del local, y las coordenadas.
- Terraza: El nodo Terraza representa una terraza específica que está asociada a un local, y se identificará por su número de terraza (num_terraza). También se incluyen atributos relacionados con su ubicación y periodo de funcionamiento.

- **Relaciones:**

- ESTÁ_EN: Se crea una relación entre el nodo Local y el nodo Barrio, indicando que el local se encuentra en un barrio determinado.
- TIENE_TERRAZA: Se crea una relación entre el nodo Local y el nodo Terraza, indicando que el local tiene una terraza asociada.

Retorno y visualización:

La consulta devuelve los nodos Barrio, Local y Terraza, limitando la salida a los primeros 100 resultados. Esto es útil para verificar cómo se construyen y conectan los nodos en el grafo.



Color Azul: Barrio

Color Naranja: Local

Color Amarillo: Terraza

Para mayor visualización del elemento grafo

Pueden descargar la imagen completa en este enlace Click [Aqui](#)

Overview

Node labels

- * (117)
- Barrio (16)
- Local (50) **Local**
- Terraza (46)
- Distrito (5)

Relationship types

- * (200)
- ESTÁ_EN (50)
- TIENE_TERRAZA (50)
- PERTENECE_A (100)**

Displaying 117 nodes, 200 relationships.

Una vez realizada la ejecución con toda la colección de datos, podemos realizar consultas sobre los datos ya creados en Neo4J

The screenshot shows the Neo4j Browser interface with the following details:

- Database Information:**
 - Use database: neo4j
 - Node labels: Local (117), Barrio (16), Distrito (5), Terraza (46), Ubicacion (3)
 - Relationship types: ESTÁ_EN (200), PERTENECE_A (100), TIENE_TERRAZA (50), TIENE_UBICACION (22)
 - Property keys: Nombre, Clase_Vial_Edificio, Comentario, Coordenada_X_Local, Coordenada_Y_Local, Desc_Barrio_Local, Desc_Distrito_Local, Desc_Periodo_Terraza, Desc_Situacion_Local, Desc_Situacion_Terraza, Desc_Ubicacion_Terraza, Id_Distrito_Local, Id_Local, Num_Edificio, Num_Terraza, Prox_Inspeccion, Puntuacion
 - Connected as: (None)
- neo4j\$ Query Results:**

```
neo4j$ MATCH (n) RETURN n LIMIT 5
```

The results show five nodes: VILLA DE VALLE, SALAMANCA, SAN BLAS, SALAMANCA, and SALAMANCA. The node labels are Local.
- neo4j\$ Call Results:**

```
neo4j$ CALL apoc.load.jsonArray('file:///locales.json') YIELD value AS locale MERGE (b: Barrio {id_distrito_local: locale.id_distrito_local}) -> (b)
```

The results show a complex network graph of relationships between nodes, primarily Local nodes.
- Overview:**

Displaying 117 nodes, 200 relationships.

 - Node labels: Local (117), Barrio (32), Terraza (121)
 - Relationship types: ESTÁ_EN (147), TIENE_TERRAZA (200), PERTENECE_A (147)

A note at the bottom states: "⚠ Not all return nodes are being displayed due to initial Node Display setting. Only first 300 nodes are displayed."