INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO

Bases de datos no relacionales

Proyecto 1

Profesor:

Dr. Felipe López Gamino

Equipo Atómico

Integrantes

Renata Claudia Monsalve Rubí – 176371

Fernando Merino Benítez – 166541

Francisco Emilio González Rico - 175575

Eduardo Woldenberg – 158279

Fecha de entrega

08/10/2019

Índice:

Introducción:

La bases de datos no relacionales ofrecen ventajas de rapidez de búsqueda y manejos de nulos, a cambio de la pérdida de una estructura bien definida y la homogeneidad de los datos encontrados para cada conjunto de datos. A diferencia de las relacionales, los conjuntos de datos no se obtienen de forma vertical mediante una clave primaria que nos regresa una tupla. En cambio, ofrece diferentes alternativas en la disposición de la información, la cual puede darse como documentos, por clusters de columnas ordenadas o por clave/valor, entre otras.

MongoDB es un ejemplo de las bases de datos no relacionales que funcionan por medio de documentos; estos no son, sin embargo, textos o archivos. Por documentos, se hace referencia a conjuntos de información débilmente estructurados, que en el caso de Mongo se presentan como diccionarios de información, donde los atributos son presentados como las claves y los datos el contenido. Cabe la pena resaltar que una clave puede dirigir a un segundo diccionario, generando así subniveles. Esto es útil cuando una clave posee más de un valor y se desea hacer una diferencia entre ambos.

Las bases de datos no relacionales resultan altamente efectivas para el manejo de grandes cantidades de información, en especial si esta cuenta con datos heterogéneos o varios nulos. Tal es el caso de la información que encontramos, por ejemplo, en la búsqueda de datos sobre las explosiones de armas nucleares que se han realizado desde que inició la carrera nuclear. Estas explosiones resultan generalmente hechas de forma secreta, puesto que representan una parte de la carrera de armamento que realizan algunas naciones como medida para prepararse para las posibles futuras guerras. Por ellos, los datos que se obtienen varían. Por otro lado, hay países que aprovechan estas explosiones como demostraciones de poder de armamento, por lo que sus datos son bastante públicos.

Más allá del contexto social e histórico de estas detonaciones, es importante conocer los datos de las explosiones debido a las repercusiones que tienen en cuanto a radioactividad, podemos realizar varios análisis mediante búsquedas que regresan los documentos que cumplen condiciones dadas, como las realizadas en un rango de tiempo, o las que tuvieron mayores dimensiones. La ubicación de las detonaciones y si fueron detonadas desde aire o bajo tierra. De estas búsquedas podemos conocer información, como cuáles son los países que podrían eventualmente desatar una guerra con armas nucleares, la década en la que entraron en la carrera o si eventualmente la dejaron, etc.

MongoDB también nos permite hacer agregamiento de datos, con lo que se pueden agrupar o filtrar las búsquedas con mayor precisión. Estas funciones nos permiten dar solución a preguntas como el número de detonaciones que se ha realizado por cada país, o el total de detonaciones realizadas en un rango espacial o temporal. Estas búsquedas, introducen la información en un *pipeline,* el cual puede contener varias capas e ir concentrando y distribuyendo la información con funciones de conteo, promedio y suma de totales entre otros. Los resultados de estas búsquedas complementan lo que se puede extraer de la base de datos sobre las detonaciones nucleares.

Solución:

Para el procesamiento de la base de datos se instaló mongoDB para hacerlo mediante documentos, por las ventajas mencionadas en la introducción. A pesar de que mongo puede ser manipulado directamente desde un command window, se utilizaron 2 herramientas para facilitar la lectura, manipulación y presentación de los datos: Compass y Pymongo.

Compass es un IDE que viene incluido en el paquete de instalación al instalar mongoDB, incluso en la versión gratuita “Community”. Nos presenta funciones útiles, como la posibilidad de importar la base desde un archivo .csv o Json, por medio de un menú y una ventana de selección. Nos permite visualizar el tamaño de la base, el número de documentos, y las colecciones que contiene cada base. Tiene un menú con entradas de texto que facilitan realizar querys sencillos y presenta los resultados de dos maneras: como texto u ordenado en una tabla donde cada documento representa una tupla (imitando las representaciones de una base de datos relacional).

Cuenta también con una ventana que facilita la construcción de funciones de agregado para crear pipelines. Permite agregar por capas las funciones de agrupamiento, conteo, matching, etc. Y aplicar a dichos resultados una segunda capa de agrupamiento. Estás funciones pueden ser guardadas en un archivo. Una última función atractiva de Compass, es la capacidad que tiene de exportar bases de datos creadas en él, así como los querys (exportables a lenguajes como Python 3). Cabe la pena mencionar que el IDE tiene varias limitaciones, por ejemplo, su llenado de querys resulta torpe para programadores experimentados en mongo, ya que no permite insertar una búsqueda de corrido: te forza a segmentarla en su tabla de inserción. Otra limitante es que no permite realizar gráficas dentro del IDE a menos que se cuente con versiones de paga de Compass.

Esta última circunstancia nos llevó a agregar Pymongo a nuestra arquitectura, como un set de herramientas que nos permite manipular las bases de datos almacenadas con una mezcla de las sintaxis de mongo y de Python. Es importante mencionar que las bases de datos existen en la conexión de localhost, creada desde Compass, así, al iniciar el código en Pymongo y declarar una nueva conexión con el mismo id (12701) y al localhost, se obtiene acceso inmediato a las bases de datos existentes. Con ello, una base creada en Compass es accesible por Pymongo y viceversa, siempre que la conexión se realice al mismo id. Aparece así la importancia de cuidar la concurrencia y de actualizar la base en Compass cuando se hagan cambios en Pymongo. Por el lado de Pymongo, cada que se corre el código se toman los datos de la base actualizada. Estos son los dos extremos que se deben considerar para la concurrencia en las bases.

El resultado es una arquitectura constituida con el siguiente diagrama:


python™
Manipulación de datos
Gráficas de resultados
mongoDB
Manipulación de la base
Visualiación de resultados y docs
MongoDB Compass

Características de la solución:

La base de datos permite que se consulte información de las pruebas nucleares. Se pueden consultar una variedad de características de estas pruebas, como, por ejemplo, el tiempo, el rendimiento del arma nuclear y el país que realizó la prueba. Además de esto se pueden juntar condiciones para conseguir resultados más concretos y específicos. Por ejemplo se puede consultar las ubicaciones (longitud y latitud) de las pruebas en un intervalo de tiempo para poder saber cómo eso pudo haber afectado ese lugar en ese tiempo. Aparte de todo eso, se puede consultar las fuentes de dónde se consiguió la información de las bombas nucleares originalmente, para poder descubrir más información de estas pruebas que pueda tener las base de datos original.

Las herramientas utilizadas fueron en su mayoría vistas en clase. Se instaló Pymongo, siguiendo las instrucciones del Dr. Felipe López, y el IDE de compass viene incluido en el paquete de descarga de mongoDB. De las herramientas no vistas en clase, que fueron usadas para este proyecto, se usó Jupyter Notebook en vez de spyder, debido a que permite segmentar el código y correrlo por secciones, lo que facilita encontrar los errores en el código y obtener los resultados de los querys de forma individual.

Para las gráficas se utilizaron librerías de Python especialmente diseñadas para graficar: pandas y matplotlib. Sin embargo, no se usaron en todo su potencial, se aplicaron únicamente para obtener visualizaciones gráficas de los resultados de los querys obtenidos en la búsqueda, pero estas librerías están diseñadas para funcionar directamente sobre tablas; sin que por ello resulten innecesarias a nuestra solución.

A continuación se muestra la parte del código donde se hace la conexión a la base de datos y las importaciones de las librerías que se requerían (y que fueron previamente instaladas):

Conexión a mongo

```
import pymongo
from pymongo import MongoClient
import matplotlib.pyplot as plt
import numpy as np
client = MongoClient('localhost', 27017)
```

Obtención y almacenamiento de los datos:

La base de datos fue obtenida usando el Dataset Search de Google. Este, es un buscador especializado para encontrar bases de datos de entre diferentes páginas y plataformas. Se decidió utilizar una base de datos que contuviera términos cuantitativos sobre cualitativos, con el objetivo de computar promedios, sumas, máximos, etc., ya que el equipo consideró que serían búsquedas más enriquecedoras que las realizadas con base en atributos cualitativos (no porque no sean relevantes, sino porque estos están sujetos a una interpretación subjetiva).

De las bases de datos arrojadas por el buscador sobre las explosiones de armas nucleares, se encontraron algunas actualizadas a la década (incluso año) actual, sin embargo, el acceso a esas bases de datos está restringido para usuarios que pagan una suscripción, misma que se hubiese pagado de no exceder el presupuesto grupal. Por ello, se optó por una base de datos gratuita que, a pesar de estar desactualizada por al menos dos décadas, conserva la información que pertenece a la época con una mayor actividad de desarrollo de armamento nuclear: la guerra fría.

Tras obtener los datos, como archivo separado por comas, se importó al programa usando Compass de mongoDB. Esta base de datos inicial resultó problemática debido a que importó todos los datos como si fuesen del tipo String. Ello impedía hacer los cálculos por los que precisamente se había escogido la base de datos. Se intentó inicialmente iterar sobre todos los documentos re asignando el tipo de datos, intento en el que se fracasó, mismo resultado que se obtuvo al intentar modificar los documentos y realizar un update para todos. Estos intentos fueron fallidos en parte por el modo en que se hibrida la sintaxis entre mongo y python, ya que mongo no recibe un string conformado por concatenaciones y ello impide que podamos anexar un índice o variable a los querys, por ello fue imposible iterar sobre todos generando un cast del tipo e ir agregando por índice.

Finalmente se optó por agregar los datos “manualmente”, es decir, conformando cada documento en un diccionario, tras recuperar el dato correspondiente desde el archivo .csv y castearlo a su tipo correspondiente. Esto se pudo realizar en todos menos las fechas, ya que estas necesariamente serían agregadas como string. Sabemos que de experimentar con código, probablemente exista alguna manera de forzar el tipo a ser Date, sin embargo, también se puede castear durante los querys, por lo que no resulta necesario hacerlo. Durante la creación de la colección, se introdujeron algunos condicionales para limpiar los datos de nulos, es decir, donde no existía información en la tabla, se omitía agregar el atributo al documento. Esto generó colecciones heterogéneas, pero el manejo de estructura débil que tienen las bases de datos no relacionales, presenta una ventaja para la manipulación de datos de este tipo.

Una vez creado el diccionario que lee y asigna los tipos de las variables, simplemente se agregaba, al final de cada iteración del ciclo, a la colección, tal como se puede ver a continuación:

Creación de base de datos y colección desde .csv

```
import csv

#Se lee el archivo
csvfile = open('nucs.csv', 'r')
reader = csv.DictReader( csvfile )
client=MongoClient()
#se crea la base
db=client.nuc_expl

#se crea la colección o se elimina en caso de existir
db.nucs.drop()

#asignación de los nombres de los atributos
header= ["_id","latitude","longitude","depth","mb","name","source","Country","datetime","max_y

#para que conserven su tipo es necesario agregarlos "manualmente", de otro moido serán
#agregados todos como strings
for each in reader:
    #Se asignaron de columna en columna para regular los tipos de datos
    row={}
    row["_id"]=int(each["_id"])
    row["latitude"]=float(each["latitude"])
    row["longitude"]=float(each["longitude"])

    #Los siguientes ifs manejan los nulos de las tablas para que no sean agregados a los docs
    if each["depth"] != 'NA':
        row["depth"]=float(each["depth"])
    if each["mb"] != 'NA':
        row["mb"]=float(each["mb"])
    if each["name"] != '-':
        row["name"]=each["name"]

    row["source"]=each["source"]
    row["Country"]=each["Country"]
    row["datetime"]= each["datetime"]
    row["max_yield"]=float(each["max_yield"])
    row["medium"]=each["medium"]
    row["confirmation"]=each["confirmation"]
    row["salvo"]=int(each["salvo"])
```

```
    #se inserta el documento recién creado
    db.nucs.insert_one(row)
```

Estructura de las "tablas" de la BD:

Descripción de las columnas:
En la base de datos hay 14 columnas diferentes: “column_a”, “latitude”, “longitude”, “location”, “depth”, “mb”, “name”, “source”, “country”, “datetime”, “max_yield”, “medium”, “confirmation” y “salvo”.

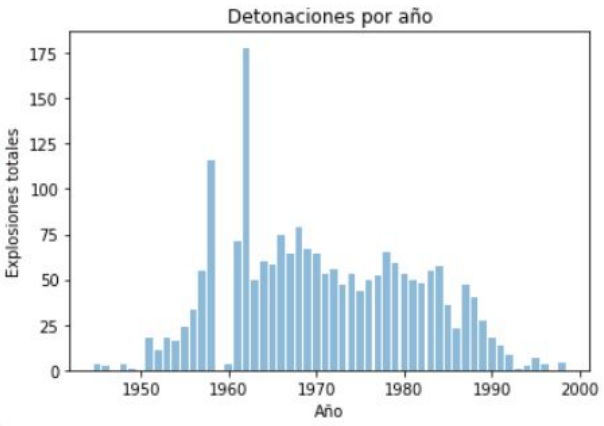
- “columna_a” es un índice para cada explosión nuclear que empieza desde 1 y sube hasta 2041.
- “latitude” y “longitude” son los valores de la latitud (en grados y norte positivo) y la longitud (en grados y oeste positivo) que definen en dónde se realizó la prueba nuclear (“location” es el conjunto de la latitud y la longitud).
- “depth” quiere decir la profundidad de la prueba y está representado en kilómetros con un valor positivo significando por debajo de la tierra.
- “mb” es el valor promedio del sesgo
- “name” es el nombre dado a la prueba de la bomba nuclear.
- “source” define el lugar de donde se consiguió la información de la explosión
- “country” es el país que realizó la prueba. Estos países incluyen China, Francia, India, Pakistán, Reino Unido, Unión Soviética y Estados Unidos. En caso que no se sabe el país se denota como “Unknown”.
- “datetime” es el día y la hora en la que se realizó la explosión.
- “max_yield” es el valor que define el rendimiento del arma nuclear y se mide en equivalente de TNT. En esta base de datos el valor significa kilotonne TNT, es decir, miles de toneladas de TNT.
- “medium” denota el medio en el cual se realizó la explosión nuclear. Esta base de datos solo considera 3 medios diferentes: aéreo (“air”), subterráneo (“underground”) y naval (“water”)
- “confirmation” significa el estado de confirmación, es decir, si la prueba es reconocida. En esta columna pueden haber 3 valores: “confirmed” (confirmado), “presumed” (supuesto) y “unkown” (desconocido)
- “salvo” es un boolean que define si la prueba se puede categorizar bajo salvo. Una prueba salvo quiere decir que se realizaron varias explosiones en un rango de 2 kilómetros y con un intervalo de tiempo menor a 0.1 segundos.

Resultados:

A continuación se presentan los 9 queries realizados con la solución. Algunos de ellos arrojan valores que pueden entenderse mejor graficados, otros arrojan un conjunto de documentos. En el caso de los segundos, para facilitar la visualización de la información resultante de la consulta, se han anexado las vistas de las tablas o nodos que se obtienen mediante Compass. Para todas las consultas se declara un pipeline, que incluye las diferentes capas de agregado después se llama con la misma línea para vaciar los resultados en una lista finalmente se grafican con métodos declarados al pie del código.

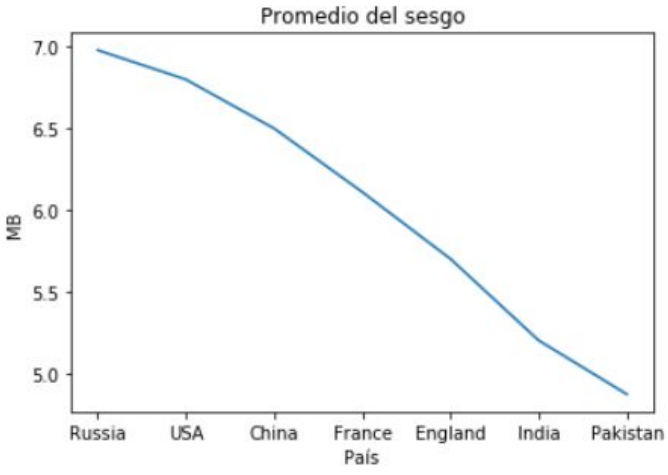
El primer query muestra las detonaciones realizadas por año, junto con su respectiva gráfica:

```
pipeline = [
    {
        '$group': {
            '_id': {
                '$year': {
                    '$dateFromString': {
                        'dateString': '$datetime'
                    }
                }
            },
            'total': {
                '$sum': 1
            }
        }
    }, {
        '$sort': {
            'total': -1
        }
    }
]
res= list(db.nucs.aggregate(pipeline))
```



La siguiente consulta muestra el sesgo promedio de las bombas agrupadas por país:

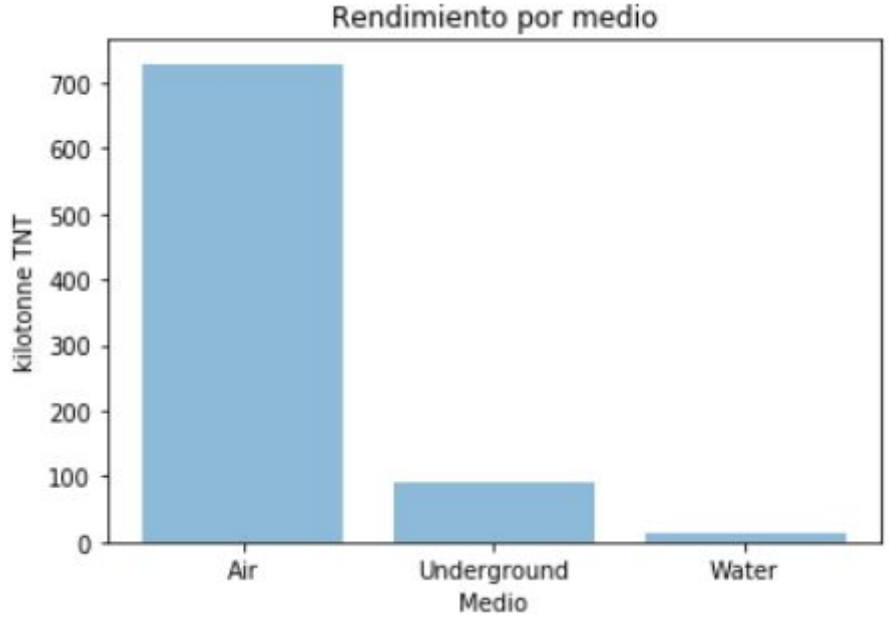
```
pipeline = [
    {
        '$group': {
            '_id': '$Country',
            'maximo': {
                '$max': '$mb'
            }
        }
    }, {
        '$sort': {
            'maximo': -1
        }
    }
]
res= list(db.nucs.aggregate(pipeline))
```



La siguiente consulta muestra el rendimiento total de las bombas dependiendo del medio en el que hayan sido detonadas, considerando que existen únicamente 3, se puede notar la información en documentos y graficada:

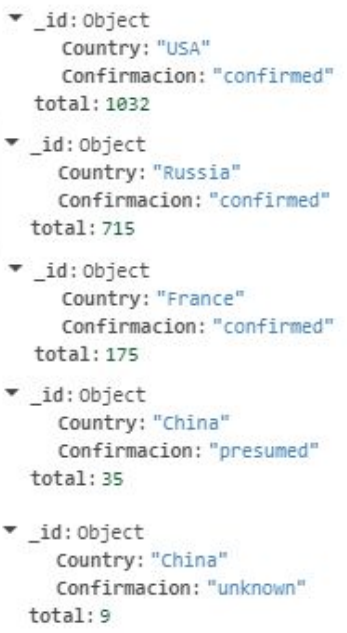
```
pipeline = [
    {
        '$group': {
            '_id': '$medium',
            'promedio': {
                '$avg': {
                    '$sum': '$max_yield'
                }
            }
        }
    }
]
res= list(db.nucs.aggregate(pipeline))
```

```
_id: "Water"
promedio: 12.830000000000002

_id: "Underground"
promedio: 90.31819478527608

_id: "Air"
promedio: 728.9125030261349
```



La siguiente consulta agrupa las detonaciones por país, y sub agrupa por estado de la confirmación (es decir, si se confirmó el lanzamiento o sólo se presupone). Los resultados se muestran desde compass, a manera de documento:

```
pipeline = [
    {
        '$group': {
            '_id': {
                'Country': '$Country',
                'Confirmacion': '$confirmation'
            },
            'total': {
                '$sum': 1
            }
        }
    }, {
        '$sort': {
            'total': -1
        }
    }
]
res= list(db.nucs.aggregate(pipeline))
```



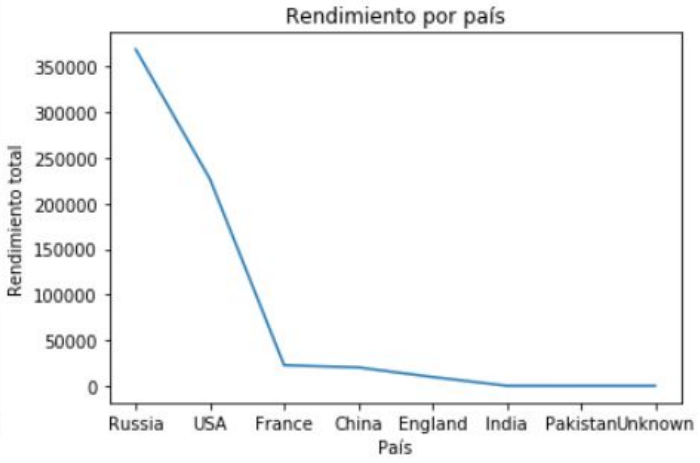
Esta consulta no se creo mediante funciones de agregado, el query fue directamente llamado desde python. Sin embargo, los documentos resultantes tienen una mejor visualización desde compass. En este ejemplo podemos ver las dos vistas: como tabla y por documentos.

```
for doc in db.nucs.find({ 'max_yield': {'$gt': 18} }, { "Country": 1, "datetime": 1 } ):
    print(doc)
```

	_id Int32	Country String	datetime String
1	1	"China"	"1964-10-16"
2	2	"China"	"1965-05-14"
3	3	"China"	"1966-05-09"
4	6	"China"	"1967-06-17"
5	9	"China"	"1969-09-22"
6	10	"China"	"1969-09-29"
7	11	"China"	"1970-10-14"
8	12	"China"	"1971-11-18"
9	15	"China"	"1973-06-27"
10	16	"China"	"1974-06-17"
11	21	"China"	"1976-11-17"
12	27	"China"	"1980-10-16"
13	33	"China"	"1987-06-05"
14	36	"China"	"1990-08-16"
15	37	"China"	"1992-05-21"
16	51	"France"	"1962-05-01"
17	52	"France"	"1963-03-18"
18	53	"France"	"1963-03-30"
19	54	"France"	"1963-10-20"
20	55	"France"	"1964-02-14"

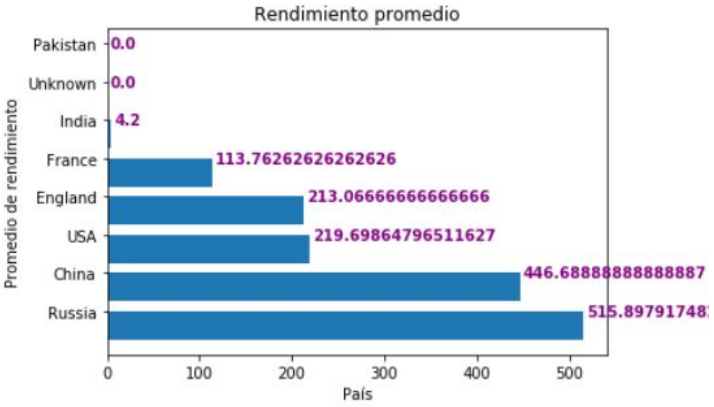
Esta consulta muestra el rendimiento total de las bombas (podemos considerar el rendimiento como la capacidad de destrucción, o la fuerza de la detonación). Rusia y EU, posiblemente, tiene valores mayores debido a que han detonado más bombas que los demás países. Por tanto, esta gráfica se puede considerar como la energía nuclear generada en total por cada país.

```
pipeline = [
    {
        '$group': {
            '_id': '$Country',
            'total_danio': {
                '$sum': '$max_yield'
            }
        }
    }, {
        '$sort': {
            'total_danio': -1
        }
    }
]
res= list(db.nucs.aggregate(pipeline))
```



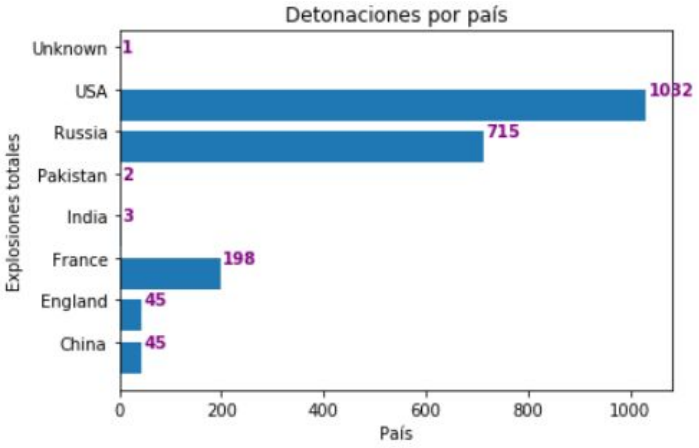
La siguiente tabla nos muestra en la gráfica, el daño promedio que producen las bombas agrupadas por país. Es decir, independientemente de la cantidad de bombas detonadas, cuál es el daño promedio que cada país ha logrado desarrollar. Es evidente que las bombas de China y Rusia son las más efectivas en liberar energía.

```
pipeline = [
    {
        '$group': {
            '_id': '$Country',
            'promedio': {
                '$avg': {
                    '$sum': '$max_yield'
                }
            }
        }
    }, {
        '$sort': {
            'promedio': -1
        }
    }
]
res= list(db.nucs.aggregate(pipeline))
```



Esta gráfica muestra el total de bombas que ha detonado cada país, como vemos, China a pesar de haber detonado pocas, ha conseguido avanzar mucho en poco tiempo, mientras que EU y Rusia comprenden, prácticamente, la totalidad de las detonaciones realizadas.

```
pipeline = [
    {
        '$group': {
            '_id': '$Country',
            'total_exp': {
                '$sum': 1
            }
        }
    }, {
        '$sort': {
            '_id': 1
        }
    }
]
res= list(db.nucs.aggregate(pipeline))
```



Finalmente, se ha agregado el código utilizado para graficar las consultas, se definieron 3 tipos de gráficas:

Funciones de graficado

```
#los 3 métodos reciben una lista, las claves para accederla y los nombres de ejes y graficas.
#de la lista se vacían los datos en arreglos paralelos
#finalmente se grafica


#gráfica de línea
def graf(lista,x,y,nomx,nomy,titulo):
    #vaciado de datos en arreglos
    varx=[]
    vary=[]
    for each in lista:
        varx.append(each[x])
        vary.append(each[y])

    #graficado
    plt.plot(varx,vary)
    plt.xlabel(nomx)
    plt.ylabel(nomy)
    plt.title(titulo)
    plt.show()


#gráfica de barras verticales
def grafBarra(lista,x,y,nomx,nomy,titulo):
    #vaciado de datos en arreglos
    varx=[]
    vary=[]
    for each in lista:
        varx.append(each[x])
        vary.append(each[y])

    #graficado
    plt.bar(varx, vary, align='center', alpha=0.5)
    plt.xlabel(nomx)
    plt.ylabel(nomy)
    plt.title(titulo)
    plt.show()
```

```
#gráfica de barras horizontales con etiquetas de valores por barra
def grafB(lista,x,y,nomx,nomy,titulo):
    #vaciado de datos en arreglos
    varx=[]
    vary=[]
    for each in lista:
        varx.append(each[x])
        vary.append(each[y])

    #graficado
    fig, ax = plt.subplots()
    width = 0.75 # ancho de las barras
    ind = np.arange(len(vary))  # la localización de las barras en x
    ax.barh(ind, vary, width)
    ax.set_yticks(ind+width/2)
    ax.set_yticklabels(varx, minor=False)
    plt.title(titulo)
    plt.xlabel(nomx)
    plt.ylabel(nomy)
    for i, v in enumerate(vary):
        ax.text(v + 3, i + .25, str(v), color='purple', fontweight='bold')
    plt.show()
```

Conclusiones:

Del trabajo realizado podemos obtener las siguientes conclusiones:

- Las bases de datos no relacionales resultan sumamente efectivas para el manejo de grandes cantidades de información.
- El manejo de los nulos que ofrecen, permiten buscar resultados sin necesidad de considerar los casos nulos, ya que estos, simplemente, no son atributos de los documentos que se tienen en la colección y, por ello, no aparecen en los resultados de las consultas.
- Las bases de datos pueden ser manipuladas mediante más de una herramienta, independientemente del tipo y del lenguaje. Estas ofrecen ventajas y desventajas que pueden ser aprovechadas al construir las consultas, considerando en cuál herramienta será más fácil realizarla o cuál de ellas me presenta la información de una forma que se acerca más a lo que deseo.
- Se pueden mezclar las capacidades de las herramientas, por ejemplo, durante la presente solución, algunas consultas se construían en compass y se ejecutaban en Pymongo, lo que permitía recuperar datos y graficar, tras haber utilizado las formas de autollenado de compass que evitan cometer errores de sintaxis.

En cuanto a la información de la base de datos seleccionada, podemos hacer otro conjunto de conclusiones:

- La carrera de armamento nuclear está dominada por 3 grandes potencias en la actualidad: Rusia, China y EU.
- A pesar de que China entró, relativamente, hace poco a la carrera armamentista, ha conseguido grandes avances en poco tiempo, lo que se vuelve palpable en la capacidad que se ha registrado de sus detonaciones.
- Han sido detonadas una cantidad importante de bombas nucleares sin que esto sea del conocimiento general de la población, ello lleva a cuestionarse cuáles son los alcances de las consecuencias de generar esta cantidad de radiación sobre la tierra.
- También invita a cuestionarnos, en qué condiciones se encuentran los lugares donde se suelen hacer las prácticas y cuantos hábitats han sido destruidos a consecuencia de necesitar lugares aislados (para mantener los resultados de las explosiones privados)

Bibliografía:

[1]X. Yang, R. North and C. Romney, *Worldwide Nuclear Explosions*. Arlington: Science Applications International Corporation. Available: https://www.ldeo.columbia.edu/~richards/my_papers/WW_nuclear_tests_IASPEI_HB.pdf [Accessed: 02- Oct- 2019].

[2]"Python MongoDB", *W3schools.com*, 2019. [Online]. Available: https://www.w3schools.com/python/python_mongodb_getstarted.asp. [Accessed: 02- Oct- 2019].

[3]"The MongoDB 4.2 Manual — MongoDB Manual", *Docs.mongodb.com*, 2019. [Online]. Available: https://docs.mongodb.com/manual/. [Accessed: 03- Oct- 2019].

[4]"Aggregation Examples — PyMongo 3.9.0 documentation", *Api.mongodb.com*, 2019. [Online]. Available: https://api.mongodb.com/python/current/examples/aggregation.html. [Accessed: 04- Oct- 2019].

[5]J. Bodnar, "PyMongo tutorial - Python MongoDB programming", *Zetcode.com*, 2019. [Online]. Available: http://zetcode.com/python/pymongo/. [Accessed: 05- Oct- 2019].