

Uso de bases de datos NoSQL

PRA1

Propuesta de solución

Instrucciones

Para poder realizar la práctica, el alumno debe usar la máquina virtual proporcionada que se encuentra en el área de recursos y consultar el manual de usuario proporcionado:

- Máquina virtual Linux Mint suministrada.
- Documento “Máquina virtual Linux Mint (Manual)”

También tenéis a vuestra disposición dos vídeos, uno de uso general de la máquina virtual y otro con sugerencias para trabajar con la base de datos de Cassandra. Aunque estéis familiarizados con las máquinas virtuales o estas bases de datos, os recomendamos su visionado, ya que podéis encontrar alguna sugerencia o idea nueva que os facilite el desarrollo de las actividades:

- Consejos sobre el uso de la máquina virtual: <https://vimeo.com/539103098>
- Consejos sobre el uso de Cassandra: <https://vimeo.com/539103072>

En algunos casos se utilizarán las bases de datos ya instaladas en la máquina virtual. En el caso de que sea necesario cargar nuevos datos, se indicará en cada ejercicio y se detallarán las instrucciones de cómo hacerlo.

Ejercicio 1: Cassandra (30%)

Los datos necesarios para este ejercicio los tendréis que cargar vosotros mismos en función del siguiente enunciado.

El sistema de gestión de reservas de una aerolínea dispone de las siguientes tablas en un sistema relacional y normalizado:



Las tablas avión y pasajero son las tablas principales, y la tabla ReservaAsiento contiene las reservas de los pasajeros para cada asiento de un determinado avión. Los datos de los que se dispone en el sistema son los siguientes:

Tabla avión:

CodigoAvion	Origen	Destino	HoraSalida	HoraLlegada
IB3001	NUEVA YORK	BARCELONA	30/03/2021 8:00:00	30/03/2021 9:30:00
IB3002	MADRID	BRUSELAS	01/07/2021 10:00:00	01/04/2021 12:30:00
IB3003	BARCELONA	NUEVA YORK	29/03/2021 7:00:00	29/03/2021 14:00:00

Tabla pasajero:

IdPasajero	Nombre	DNI
101	JORGE ANTONIO GARCIA	111A
102	GEORGE DE BROUCKERE	111C
104	MARIA MESTRE	111E
105	RODRIGO CALVO	111F
106	ORIOL MENEZES	111G
107	CARLO BERRUZO	111H
108	SOFIA CANYADELL	111I
109	ARIANNA RUIZ	111K

110	NAIARA ZAPICO	111L
-----	---------------	------

Tabla ReservaAsiento:

CodigoAsiento	CodigoAvion	IdPasajero
1A	IB3001	101
2B	IB3001	104
2F	IB3001	105
3A	IB3001	106
3B	IB3001	107
3C	IB3001	108
4B	IB3001	109
1A	IB3002	102
1C	IB3002	105
2A	IB3002	110
1A	IB3003	101
1C	IB3003	102
5A	IB3003	105
7F	IB3003	106
9D	IB3003	107

Lo primero que deberéis realizar es conectaros a cassandra y crear un keyspace donde ubicar las familias como por ejemplo:

```
create keyspace aerolinea with replication ={ 'class':  
'SimpleStrategy','replication_factor': 1};
```

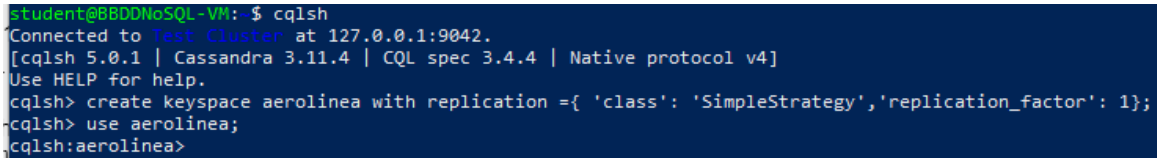
Y después utilizar el keyspace creado:

```
use aerolinea;
```

De manera que obtendremos:

```
cqlsh> create keyspace aerolinea with replication ={ 'class':
'SimpleStrategy','replication_factor': 1};
cqlsh> use aerolinea;
cqlsh:aerolinea>
```

Para justificar la ejecución de los comandos en los ejercicios se puede adjuntar la salida de la consola de comandos o una captura de pantalla como la siguiente:



```
student@88DDNoSQL-VM:~$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.4 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> create keyspace aerolinea with replication ={ 'class': 'SimpleStrategy','replication_factor': 1};
cqlsh> use aerolinea;
cqlsh:aerolinea>
```

Lo que se pretende con este ejemplo es mostrar el formato de entrega de vuestra actividad, además de que podáis aprovechar la creación del keyspace que es totalmente válido para esta actividad. Notad que tanto en el texto copiado de la consola como en la captura de pantalla, se ve la ejecución del comando y el resultado de la ejecución del mismo: `cqlsh:aerolinea>` indica que ya estamos utilizando el keyspace creado.

Si el comando que estáis utilizando tiene alguna salida de texto (como por ejemplo las consultas) deberéis adjuntarla en texto o captura de pantalla.

El keyspace creado anteriormente es a modo de ejemplo. Tenéis total libertad de utilizar otro nombre o configuración si lo preferís, aunque no será un aspecto a valorar en el bloque de Cassandra.

Ejercicio 1.1 (no puntúa):

Una vez creado el keyspace, deberéis crear dos familias de columnas (tablas en Cassandra) y rellenarlas con los datos detallados anteriormente. Las familias de columnas deberán permitir ejecutar las consultas indicadas en el ejercicio de la siguiente forma:

1. La primera debe permitir recuperar la información de **vuelos y sus reservas (VueloReservas)** para las consultas 1 a 4.
2. La segunda debe permitir recuperar la información de **pasajeros y sus reservas (PasajeroReservas)** para las consultas 5 y 6.

Se debe:

- Ejecutar el código CQL para crear cada familia de columnas (tablas) solicitadas anteriormente.
- Ejecutar las sentencias CQL para cada fila que tengáis que insertar en la familia de columnas.

Esta parte es guiada (por lo tanto no puntúa) pero es necesaria para la ejecución de las consultas del siguiente ejercicio.

Utilizaremos el keyspace especificado en el ejemplo.

Crearemos la familia de columnas (o tabla) VueloReservas con la siguiente instrucción CQL:

```
CREATE TABLE VueloReservas (CodigoAvion TEXT, Origen TEXT, Destino
TEXT, CodigoAsiento TEXT, Nombre TEXT, DNI TEXT, Primary
key(codigoavion, codigoasiento));
cqlsh:aerolinea>
```

Después, insertamos los datos con las siguientes instrucciones CQL:

```
insert into VueloReservas (CodigoAvion, Origen, Destino,
CodigoAsiento, Nombre, DNI) values ('IB3002','MADRID', 'BRUSELAS',
'2A', 'NAIARA ZAPICO', '111L');
```

```
insert into VueloReservas (CodigoAvion, Origen, Destino,
CodigoAsiento, Nombre, DNI) values ('IB3003','BARCELONA', 'NUEVA
YORK', '1A', 'JORGE ANTONIO GARCIA', '111A');
```

```
insert into VueloReservas (CodigoAvion, Origen, Destino,
CodigoAsiento, Nombre, DNI) values ('IB3003','BARCELONA', 'NUEVA
YORK', '1C', 'GEORGE DE BROUCKERE', '111C');
```

```
insert into VueloReservas (CodigoAvion, Origen, Destino,
CodigoAsiento, Nombre, DNI) values ('IB3003','BARCELONA', 'NUEVA
YORK', '5A', 'RODRIGO CALVO', '111F');
```

```
insert into VueloReservas (CodigoAvion, Origen, Destino,  
CodigoAsiento, Nombre, DNI) values ('IB3003','BARCELONA', 'NUEVA  
YORK', '7F', 'ORIOI MENEZES', '111G');
```

```
insert into VueloReservas (CodigoAvion, Origen, Destino,  
CodigoAsiento, Nombre, DNI) values ('IB3003','BARCELONA', 'NUEVA  
YORK', '9D', 'CARLO BERRUZO', '111H');
```

```
insert into VueloReservas (CodigoAvion, Origen, Destino,  
CodigoAsiento, Nombre, DNI) values ('IB3001','NUEVA YORK',  
'BARCELONA', '1A', 'JORGE ANTONIO GARCIA', '111A');
```

```
insert into VueloReservas (CodigoAvion, Origen, Destino,  
CodigoAsiento, Nombre, DNI) values ('IB3001','NUEVA YORK',  
'BARCELONA', '2B', 'MARIA MESTRE', '111E');
```

```
insert into VueloReservas (CodigoAvion, Origen, Destino,  
CodigoAsiento, Nombre, DNI) values ('IB3001','NUEVA YORK',  
'BARCELONA', '2F', 'RODRIGO CALVO', '111F');
```

```
insert into VueloReservas (CodigoAvion, Origen, Destino,  
CodigoAsiento, Nombre, DNI) values ('IB3001','NUEVA YORK',  
'BARCELONA', '3A', 'ORIOI MENEZES', '111G');
```

```
insert into VueloReservas (CodigoAvion, Origen, Destino,  
CodigoAsiento, Nombre, DNI) values ('IB3001','NUEVA YORK',  
'BARCELONA', '3B', 'CARLO BERRUZO', '111H');
```

```
insert into VueloReservas (CodigoAvion, Origen, Destino,  
CodigoAsiento, Nombre, DNI) values ('IB3001','NUEVA YORK',  
'BARCELONA', '3C', 'SOFIA CANYADELL', '111I');
```

```
insert into VueloReservas (CodigoAvion, Origen, Destino,  
CodigoAsiento, Nombre, DNI) values ('IB3001','NUEVA YORK',  
'BARCELONA', '4B', 'ARIANNA RUIZ', '111K');
```

```
insert into VueloReservas (CodigoAvion, Origen, Destino,  
CodigoAsiento, Nombre, DNI) values ('IB3002','MADRID', 'BRUSELAS',  
'1A', 'GEORGE DE BROUCKERE', '111C');
```

```
insert into VueloReservas (CodigoAvion, Origen, Destino,  
CodigoAsiento, Nombre, DNI) values ('IB3002','MADRID', 'BRUSELAS',  
'1C', 'RODRIGO CALVO', '111F');
```


Crearemos la familia de columnas (o tabla) PasajeroReservas con la instrucción CQL que tendrá las mismas columnas, pero diferente clave primaria, por lo que agrupará y nos permitirá consultar los datos de diferente manera.

```
CREATE TABLE PasajeroReservas (CodigoAvion TEXT, Origen TEXT,
Destino TEXT, CodigoAsiento TEXT, Nombre TEXT, DNI TEXT, Primary
key(Nombre, CodigoAvion));
cqlsh:aerolinea>
```

Insertamos los datos con las siguientes instrucciones CQL:

```
insert into PasajeroReservas (CodigoAvion, Origen, Destino,
CodigoAsiento, Nombre, DNI) values ('IB3002','MADRID', 'BRUSELAS',
'2A', 'NAIARA ZAPICO', '111L');
```

```
insert into PasajeroReservas (CodigoAvion, Origen, Destino,
CodigoAsiento, Nombre, DNI) values ('IB3003','BARCELONA', 'NUEVA
YORK', '1A', 'JORGE ANTONIO GARCIA', '111A');
```

```
insert into PasajeroReservas (CodigoAvion, Origen, Destino,
CodigoAsiento, Nombre, DNI) values ('IB3003','BARCELONA', 'NUEVA
YORK', '1C', 'GEORGE DE BROUCKERE', '111C');
```

```
insert into PasajeroReservas (CodigoAvion, Origen, Destino,
CodigoAsiento, Nombre, DNI) values ('IB3003','BARCELONA', 'NUEVA
YORK', '5A', 'RODRIGO CALVO', '111F');
```

```
insert into PasajeroReservas (CodigoAvion, Origen, Destino,
CodigoAsiento, Nombre, DNI) values ('IB3003','BARCELONA', 'NUEVA
YORK', '7F', 'ORIOLE MENEZES', '111G');
```

```
insert into PasajeroReservas (CodigoAvion, Origen, Destino,
CodigoAsiento, Nombre, DNI) values ('IB3003','BARCELONA', 'NUEVA
YORK', '9D', 'CARLO BERRUZO', '111H');
```

```
insert into PasajeroReservas (CodigoAvion, Origen, Destino,
CodigoAsiento, Nombre, DNI) values ('IB3001','NUEVA YORK',
'BARCELONA', '1A', 'JORGE ANTONIO GARCIA', '111A');
```

```
insert into PasajeroReservas (CodigoAvion, Origen, Destino,
CodigoAsiento, Nombre, DNI) values ('IB3001','NUEVA YORK',
'BARCELONA', '2B', 'MARIA MESTRE', '111E');
```

```
insert into PasajeroReservas (CodigoAvion, Origen, Destino,
CodigoAsiento, Nombre, DNI) values ('IB3001','NUEVA YORK',
'BARCELONA', '2F', 'RODRIGO CALVO', '111F');
```

```
insert into PasajeroReservas (CodigoAvion, Origen, Destino,  
CodigoAsiento, Nombre, DNI) values ('IB3001','NUEVA YORK',  
'BARCELONA', '3A', 'ORIOI MENEZES', '111G');
```

```
insert into PasajeroReservas (CodigoAvion, Origen, Destino,  
CodigoAsiento, Nombre, DNI) values ('IB3001','NUEVA YORK',  
'BARCELONA', '3B', 'CARLO BERRUZO', '111H');
```

```
insert into PasajeroReservas (CodigoAvion, Origen, Destino,  
CodigoAsiento, Nombre, DNI) values ('IB3001','NUEVA YORK',  
'BARCELONA', '3C', 'SOFIA CANYADELL', '111I');
```

```
insert into PasajeroReservas (CodigoAvion, Origen, Destino,  
CodigoAsiento, Nombre, DNI) values ('IB3001','NUEVA YORK',  
'BARCELONA', '4B', 'ARIANNA RUIZ', '111K');
```

```
insert into PasajeroReservas (CodigoAvion, Origen, Destino,  
CodigoAsiento, Nombre, DNI) values ('IB3002','MADRID', 'BRUSELAS',  
'1A', 'GEORGE DE BROUCKERE', '111C');
```

```
insert into PasajeroReservas (CodigoAvion, Origen, Destino,  
CodigoAsiento, Nombre, DNI) values ('IB3002','MADRID', 'BRUSELAS',  
'1C', 'RODRIGO CALVO', '111F');
```

Ejercicio 1.2 (75%):

Con los datos que habéis cargado en el apartado anterior, se requiere realizar las siguientes consultas y adjuntar su resultado.

Limitaciones para las consultas:

- No se puede buscar manualmente el código de vuelo o el identificador de una persona en la base de datos. Se deben recuperar los datos solicitados a partir de la información facilitada en el enunciado. En un escenario real con muchos registros no es viable buscar manualmente los datos.
- No se puede utilizar la cláusula ALLOW FILTERING.

Nota: puede que algunas consultas retornen error y no podáis ejecutarlas directamente. En tal caso, debéis adjuntar las consultas que os dan error y después los comandos que habéis utilizado para resolver este error. **Se valorará todo el procedimiento que habéis seguido para poder ejecutar las consultas y obtener la información que se solicita** especialmente para aquellas consultas que no son directas.

Consulta 1 (10%):

El personal del aeropuerto de origen necesita tener un listado de todas las reservas para preparar los asientos del avión código IB3001. El listado debe mostrar: código de avión, origen, destino, y los códigos de los asientos ocupados. No se necesita información concreta del pasajero.

Consulta 2 (10%):

El departamento de marketing necesita saber cual es la ocupación de los aviones. Por lo que solicita un listado de **todos los aviones** con un recuento de los asientos reservados en cada uno. El listado debe mostrar: código de avión, origen, destino y el recuento de los asientos reservados. Sugerencia: podéis modificar la consulta anterior para realizar este ejercicio.

Consulta 3 (20%):

El departamento comercial quiere promocionar los vuelos con destino a Nueva York y quiere saber en qué estado se encuentra la ocupación del avión (o aviones) que vuelan a Nueva York. Para ello necesita un listado con las mismas columnas que la anterior consulta, pero esta vez solo para aquellos aviones con destino Nueva York. Sugerencia: podéis modificar la consulta anterior para realizar este ejercicio aunque puede que la solución no sea tan directa.

Consulta 4 (15%):

El personal del control de salida quiere tener el listado de los pasajeros con sus asientos del vuelo con código IB3001. Se necesita nombre, DNI y código de asiento ordenado por código de asiento **descendente** ya que el embarque desde atrás hacia adelante resulta más ágil.

Consulta 5 (15%):

El personal de márketing quiere premiar a los pasajeros que más viajan en sus aerolíneas. Se necesita un listado de pasajeros con un recuento de todas sus reservas. El listado debe tener el nombre del pasajero y el recuento de reservas.

Consulta 6 (30%):

El pasajero JORGE ANTONIO GARCIA quiere cambiar el asiento del vuelo IB3001 al asiento 4C. Podemos asumir que el asiento está libre, no es necesario realizar ninguna comprobación adicional como parte del ejercicio. Si no hay salida de texto en una consulta de actualización de datos, podéis ejecutar una select al final del ejercicio para mostrar que los datos han sido modificados correctamente.

SOLUCIÓN:

Una vez creadas las familias de columnas podemos ejecutar las consultas:

Consulta 1:

```
SELECT CodigoAvion, origen, destino, codigoAsiento
FROM VueloReservas
WHERE CodigoAvion = 'IB3001';
```

Salida:

codigoavion	origen	destino	codigoasiento
IB3001	NUEVA YORK	BARCELONA	1A
IB3001	NUEVA YORK	BARCELONA	2B
IB3001	NUEVA YORK	BARCELONA	2F
IB3001	NUEVA YORK	BARCELONA	3A
IB3001	NUEVA YORK	BARCELONA	3B
IB3001	NUEVA YORK	BARCELONA	3C
IB3001	NUEVA YORK	BARCELONA	4B

(7 rows)

cqlsh:aerolinea>

Consulta 2:

```
SELECT CodigoAvion, origen, destino, count(codigoAsiento)
FROM VueloReservas
GROUP BY CodigoAvion;
```

Retorna:

codigoavion	origen	destino	system.count(codigoasiento)
IB3001	NUEVA YORK	BARCELONA	7

IB3003		BARCELONA		NUEVA YORK		5
IB3001		NUEVA YORK		BARCELONA		7
IB3002		MADRID		BRUSELAS		3

(3 rows)

Warnings :

Aggregation query used without partition key

La advertencia *Aggregation query used without partition key* indica que la consulta, potencialmente, no podrá ser resuelta localmente en un nodo del cluster y probablemente implica consultar los datos de todos los nodos del cluster. Esto hace que potencialmente tenga un tiempo largo de ejecución, por lo que podríamos obtener *timeouts* si ejecutamos la consulta en una base de datos con un elevado volumen de datos (algo bastante habitual en las bases de datos NoSQL). Se trata de una consulta potencialmente costosa.

Todas las consultas que tardan tiempo en ser procesadas saturan el servidor consumiendo muchos recursos. Y el problema se puede agravar todavía más si se tiene que ejecutar muy a menudo o concurrentemente con otras consultas. Por este motivo, Cassandra añade esta advertencia en el resultado si detecta que una consulta no utiliza todas las claves o índices que debería.

[Aquí encontraréis más información](#) de manera breve (y en mi opinión) acertada.

Consulta 3:

Si ejecutamos la consulta directamente:

```
SELECT CodigoAvion, origen, destino, count(codigoAsiento)
FROM VueloReservas
WHERE destino = 'NUEVA YORK'
GROUP BY CodigoAvion;
```

Obtendremos el error:

```
InvalidRequest: Error from server: code=2200 [Invalid query]
message="Cannot execute this query as it might involve data filtering
and thus may have unpredictable performance. If you want to execute
this query despite the performance unpredictability, use ALLOW
FILTERING"
```

Como la cláusula `ALLOW FILTERING` no está permitida, tendremos como opción incluir la ciudad de destino como clave primaria, pero no nos interesa porque:

- Influirá cómo se guardan físicamente los datos de la tabla.
- Tendremos que volver a crear la tabla y volver a cargar sus datos.

Otra opción es crear un índice para esta columna que nos permitirá filtrar por la misma sin cambios en la estructura de almacenamiento y sin penalización en el rendimiento cuando ejecutemos la consulta. Además se trata de una sola consulta que requiere este filtro en esta tabla. Por lo tanto, creamos un índice:

```
CREATE INDEX idx_VueloReservas_destino ON VueloReservas(destino);
```

Y posteriormente, ejecutamos la misma consulta obteniendo (en la captura se puede ver el resultado de la creación del índice y el resultado de la consulta):

```
cqlsh:aerolinea> CREATE INDEX idx_VueloReservas_destino ON VueloReservas(destino);
cqlsh:aerolinea> SELECT CodigoAvion, origen, destino, count(codigoAsiento)
... FROM VueloReservas
... WHERE destino = 'NUEVA YORK'
... GROUP BY CodigoAvion;

codigoavion | origen | destino | system.count(codigoasiento)
-----+-----+-----+-----
      IB3003 | BARCELONA | NUEVA YORK | 5

(1 rows)

Warnings :
Aggregation query used without partition key

cqlsh:aerolinea> 
```

Consulta 4:

La consulta 4 funcionará directamente si hemos definido la clave primaria con las columnas codigoavion y codigoasiento.

```
select Nombre, DNI, codigoasiento
FROM VueloReservas
WHERE codigoavion = 'IB3001'
ORDER BY codigoAsiento DESC;
```

Dará como resultado:

nombre	dni	codigoasiento
ARIANNA RUIZ	111K	4B
SOFIA CANYADELL	111I	3C
CARLO BERRUZO	111H	3B
ORIOI MENEZES	111G	3A
RODRIGO CALVO	111F	2F
MARIA MESTRE	111E	2B
JORGE ANTONIO GARCIA	111A	1A

(7 rows)

Consulta 5:

Para esta consulta utilizaremos la tabla o familia PasajeroReservas. Si hemos elegido bien la clave primaria es directa:

```
select nombre, count(codigoasiento)
FROM PasajeroReservas
group by nombre;
```

Dará como resultado:

nombre	system.count(codigoasiento)
JORGE ANTONIO GARCIA	2
ARIANNA RUIZ	1
NAIARA ZAPICO	1
RODRIGO CALVO	3
ORIOI MENEZES	2
SOFIA CANYADELL	1
CARLO BERRUZO	2
GEORGE DE BROUCKERE	2
MARIA MESTRE	1

(9 rows)

Warnings :

Aggregation query used without partition key

cqlsh:aerolinea>

Consulta 6:

La información está replicada en dos tablas, por lo tanto tendremos que modificarla en ambas. Teniendo en cuenta la elección de las claves primarias, puede que el procedimiento sea diferente para cada tabla.

Para realizar un update se deben indicar en el where las columnas que forman parte de la clave primaria. Por lo tanto, primero debemos recuperar los valores que necesitaremos para poder ejecutarlo. Notar que el `select *` no es una buena práctica en entornos de producción, aunque no hay problema en entornos de desarrollo o pruebas.

```
select * from Vueloreservas where Nombre = 'JORGE ANTONIO GARCIA';
```

Nos retorna:

```
InvalidRequest: Error from server: code=2200 [Invalid query]
message="Cannot execute this query as it might involve data filtering
and thus may have unpredictable performance. If you want to execute
this query despite the performance unpredictability, use ALLOW
FILTERING"
```

Como se ha hecho anteriormente creamos un índice para poder consultar por nombre:

```
CREATE INDEX idx_VueloReservas_Nombre ON VueloReservas(Nombre);
```

Y ahora sí, obtenemos los datos necesarios para intentar hacer un update

```
select * from Vueloreservas where Nombre = 'JORGE ANTONIO GARCIA'
AND codigoAvion = 'IB3001';
```

```

codigoavion | codigoasiento | destino | dni | nombre | origen
-----+-----+-----+-----+-----+-----
IB3001 | 1A | BARCELONA | 111A | JORGE ANTONIO GARCIA | NUEVA YORK

(1 rows)
cqlsh:aerolinea>
```

Intentamos actualizar:

```
update VueloReservas
set CodigoAsiento = '4C'
where codigoavion = 'IB3001' AND codigoasiento = '1A';
```

Pero no funciona porque:

```
InvalidRequest: Error from server: code=2200 [Invalid query]
message="PRIMARY KEY part codigoasiento found in SET part"
```

Por lo tanto, no nos queda otro remedio que insertar una nueva fila y borrar la antigua.

```
insert into VueloReservas (CodigoAvion, Origen, Destino,
CodigoAsiento, Nombre, DNI) values ('IB3001','NUEVA YORK',
'BARCELONA', '4C', 'JORGE ANTONIO GARCIA', '111A');
```

```
delete from VueloReservas
where codigoavion = 'IB3001' AND codigoasiento = '1A';
```

Comprobamos:

```
select * from Vueloreservas where Nombre = 'JORGE ANTONIO GARCIA'
AND codigoAvion = 'IB3001';
```



```

codigoavion | codigoasiento | destino | dni | nombre | origen
-----+-----+-----+-----+-----+-----
IB3001 | 4C | BARCELONA | 111A | JORGE ANTONIO GARCIA | NUEVA YORK

```

```

(1 rows)
cqlsh:aerolinea>

```

Finalmente (y opcionalmente) borramos el índice que hemos creado para esta operación:

```
DROP INDEX idx_VueloReservas_Nombre;
```

Nos queda actualizar la segunda tabla:

Intentamos obtener los datos necesarios para el update con la consulta:

```
SELECT * FROM PasajeroReservas WHERE NOMBRE = 'JORGE ANTONIO GARCIA'
AND codigoAvion = 'IB3001';
```

Afortunadamente no es necesario crear ningún índice adicional, retorna:

```
cqlsh:aerolinea> SELECT * FROM PasajeroReservas WHERE NOMBRE = 'JORGE ANTONIO
GARCIA' AND codigoAvion = 'IB3001';
```

```

nombre | codigoavion | codigoasiento | destino | dni | origen
-----+-----+-----+-----+-----+-----
JORGE ANTONIO GARCIA | IB3001 | 1A | BARCELONA | 111A | NUEVA YORK

```

```

(1 rows)
cqlsh:aerolinea>

```

Intentamos el update directo:

```

update PasajeroReservas
set CodigoAsiento = '4C'
WHERE NOMBRE = 'JORGE ANTONIO GARCIA' AND codigoAvion = 'IB3001';

```

Que funciona sin problemas porque el campo que actualizamos no forma parte de la clave primaria y podemos especificar todas las columnas en el where. Comprobamos el resultado ejecutando la misma consulta que hemos ejecutado anteriormente y podemos ver como los datos están actualizados correctamente:

```
SELECT * FROM PasajeroReservas WHERE NOMBRE = 'JORGE ANTONIO GARCIA' AND codigoAvion
= 'IB3001';
```

```

nombre | codigoavion | codigoasiento | destino | dni | origen
-----+-----+-----+-----+-----+-----
JORGE ANTONIO GARCIA | IB3001 | 4C | BARCELONA | 111A | NUEVA YORK

```

```

(1 rows)
cqlsh:aerolinea>

```

Ejercicio 1.3 (25%)

¿Por qué no se ha elegido la combinación de las dos columnas “Nombre” y “código asiento” como clave primaria en la tabla PasajeroReservas? Justifica la respuesta con datos de ejemplo.

SOLUCIÓN:

No se ha elegido la combinación de Nombre y código asiento porque no garantiza la unicidad de la clave primaria. **De hecho hay un pasajero que tiene el mismo código de asiento asignado en dos vuelos diferentes**, lo cual causaría que solamente podríamos insertar una de las dos filas (en cada inserción sobre-escribiríamos el valor anterior quedando solamente el último insert, ya que el insert es en realidad un upsert).

A modo de ejemplo, se puede ver como el pasajero JORGE ANTONIO GARCIA tiene el mismo asiento (1A) en dos vuelos. Por lo tanto, si utilizamos el nombre y el asiento como clave, solamente se podría insertar una reserva:

IB3003	BARCELONA	NUEVA YORK	1A	JORGE ANTONIO GARCIA	111A
IB3001	NUEVA YORK	BARCELONA	1A	JORGE ANTONIO GARCIA	111A

Si ejecutamos el insert de las dos filas, la segunda fila sobre escribirá la primera.

Aunque no forma parte de la pregunta y solo es un comentario añadido, para la tabla VueloReservas se ha elegido la combinación de columnas codigoavion, codigoasiento porque garantizan la unicidad de la clave primaria y permiten realizar las consultas de forma directa y óptima (de la misma manera que para la tabla PasajeroReservas).

Ejercicio 2: Neo4j (30 %)

Considera la base de datos del caso de ejemplo de Twitter tratado en el tema 12, que describe la implementación de una base de datos en Neo4j. También se necesitará la máquina virtual de LinuxMint (que encontraréis en los recursos del aula) que contiene una instalación de Neo4j con la base de datos descrita anteriormente ya cargada. El manual para utilizar e instalar la máquina virtual también está en los recursos del aula.

Se pide proporcionar las siguientes consultas en Cypher y una captura de pantalla con los resultados que se obtienen para las siguientes operaciones:

Consulta 1 (20%): Buscar los usuarios relevantes que viven en Barcelona y que han escrito tweets que contienen la palabra “Malware” en su texto. Se debe retornar el nombre del usuario y el texto completo del tweet.

Propuesta de solución

Neo4j es sensible a mayúsculas y minúsculas, por lo que hay tres soluciones posibles dependiendo del texto utilizado en el predicado. Si buscamos “Malware” con la M en mayúsculas la consulta y la solución serían:

```
MATCH (u:RelevantTwitterUser{location:"Barcelona"})-[HAS_WRITEN]->
(r:Tweet)
WHERE r.text CONTAINS "Malware"
RETURN u.userName, r.text;
```



```
neo4j> MATCH (u:RelevantTwitterUser{location:"Barcelona"})-[HAS_WRITEN]->(r:Tweet)
        WHERE r.text CONTAINS "Malware"
        RETURN u.userName, r.text;
+-----+-----+
| u.userName | r.text |
+-----+-----+
| "Android en espa" | "Malware en #android http://bit.ly/jz9A0y Los permisos deber...
n ser más granulados así se podría evitar que accedieran a todo" |
| "Jordi Serra" | "RT @ConexionInversa: Malware de antes para los Mac OSX de aho
ra: http://t.co/ltVVI6p" |
+-----+-----+
2 rows available after 11 ms, consumed after another 108 ms
neo4j>
```

En caso que hayáis decidido buscar el texto comenzando la palabra “malware” con la “m” en minúsculas la consulta y el resultado serían:

```
neo4j> MATCH (u:RelevantTwitterUser{location:"Barcelona"})-
[HAS_WRITEN]->
    (r:Tweet)
    WHERE r.text CONTAINS "malware"
    RETURN u.userName, r.text;
```

```
+-----+
| u.userName | r.text |
+-----+
| "Android en espa" | "Nuevo malware en Android, ahora desde los banners http://bit.ly/19vRqV" |
| "Android en espa" | "El debate alrededor de los virus, malware de #Android est caliente Qu opinas t? http://bit.ly/jbzcKP" |
| "Android en espa" | "#Androides M malware para Android: Plankton, YZHCMS y DroidKungFu http://bit.ly/ihT7FO" |
| "Android en espa" | "No tengas miedo de los virus, malware, spyware :) @Lookout te mantiene seguro http://t.co/EN9c4V5" |
| "Android en espa" | "Nuevo Artículo: DroidDreamLight, nuevo malware de los creadores de DroidDream: Al principio http://goo.gl/fb/19qDO" |
+-----+
```

En caso que hayáis decidido buscar por las dos posibilidades, la “M” y la “m”, la consulta y resultado serían:

```
neo4j> MATCH (u:RelevantTwitterUser{location:"Barcelona"})-
[HAS_WRITEN]->
    (r:Tweet)
    WHERE r.text CONTAINS "malware" OR r.text CONTAINS "Malware"
    RETURN u.userName, r.text;
```

```
+-----+
| u.userName | r.text |
+-----+
| "Android en espa" | "Nuevo malware en Android, ahora desde los banners http://bit.ly/19vRqV" |
| "Android en espa" | "El debate alrededor de los virus, malware de #Android est caliente Qu opinas t? http://bit.ly/jbzcKP" |
| "Android en espa" | "Malware en #android http://bit.ly/jz9A0y Los permisos deber an ser m granulados as se podra evitar que accedieran a todo" |
| "Android en espa" | "#Androides M malware para Android: Plankton, YZHCMS y DroidKungFu http://bit.ly/ihT7FO" |
| "Android en espa" | "No tengas miedo de los virus, malware, spyware :) @Lookout te mantiene seguro http://t.co/EN9c4V5" |
| "Android en espa" | "Nuevo Artículo: DroidDreamLight, nuevo malware de los creadores de DroidDream: Al principio http://goo.gl/fb/19qDO" |
| "Jordi Serra" | "RT @ConexionInversa: Malware de antes para los Mac OSX de ahora: http://t.co/ltVVI6p" |
+-----+
```

Las tres posibilidades se consideran correctas para esta práctica.

Consulta 2 (25%): Obtener el nombre de la localización desde la que se han escrito exactamente 26 respuestas. Mostrar el nombre de la localización y el número de respuestas (que deberá ser 26).

Propuesta de solución

```
MATCH (l:Location)<-[w:IS_WRITEN_FROM]-(r:Reply)
WITH l.name as location , count(w) AS numReplies
WHERE numReplies=26
RETURN location , numReplies LIMIT 1;
```

La cláusula LIMIT es opcional.

```
neo4j> MATCH (l:Location)-[w:IS_WRITEN_FROM]-(r:Reply)
        WITH l.name as location , count(w) AS numReplies
        WHERE numReplies=26
        RETURN location , numReplies LIMIT 1;
+-----+
| location | numReplies |
+-----+
| "Armillà" | 26         |
+-----+

1 row available after 59 ms, consumed after another 0 ms
neo4j> 
```

Consulta 3 (25%): Obtener el usuario relevante con idioma de perfil codificado como 'en' que más tweets ha escrito. Se debe mostrar el nombre de usuario y el número de tweets enviados.

Propuesta de solución

```
MATCH (t:Tweet)-[:HAS_WRITEN]-(r:RelevantTwitterUser)-
[:HAS_AS_PROFILE_LANGUAGE]->(:Language {languageCode:'en'})
WITH r,COUNT(t) AS escritos
RETURN r.userName, escritos
ORDER BY escritos DESC LIMIT 1;
```

```
neo4j> MATCH (t:Tweet)-[:HAS_WRITEN]-(r:RelevantTwitterUser)-
        [:HAS_AS_PROFILE_LANGUAGE]->(:Language {languageCode:'en'})
        WITH r,COUNT(t) AS num_tweets
        RETURN r.userName, num_tweets
        ORDER BY num_tweets DESC LIMIT 1;
+-----+
| r.userName | num_tweets |
+-----+
| "Kamen"    | 5293       |
+-----+

1 row available after 180 ms, consumed after another 0 ms
neo4j> 
```

Consulta 4 (30%): Con una única consulta en cypher, calcular la ratio obtenida realizando la división con decimales A/B, siendo A y B los valores siguientes:

- **A:** Número de tweets que son respuestas de tweets geolocalizados que han sido escritos desde las ubicaciones de "Liverpool", "Seattle" u "Oxford".
- **B:** Número de tweets que son respuestas de tweets geolocalizados desde las ubicaciones de "Madrid", "Valencia" u "Sevilla".

Propuesta de solución

```

MATCH (et:Tweet)-[:IS_A_REPLY_OF]->(r:GeoLocatedTweet)-
    [:IS_WRITEN_FROM]->(l:Location)
WHERE l.name IN ["Liverpool","Seattle","Oxford"]
WITH count(et) AS englishReplies

MATCH (st:Tweet)-[:IS_A_REPLY_OF]->(r:GeoLocatedTweet)-
    [:IS_WRITEN_FROM]->(l:Location)
WHERE l.name IN ["Madrid","Valencia","Sevilla"]
WITH count(st) AS spanishReplies, englishReplies AS englishReplies

RETURN (englishReplies)*1.0/spanishReplies AS ratio

```

Con el fin de que el resultado final corresponda a un número decimal, se multiplica por 1.0 debido a que cuando numerador y denominador son valores enteros, como el caso en cuestión, se realiza por defecto la división entera.

```

neo4j> MATCH (et:Tweet)-[:IS_A_REPLY_OF]->(r:GeoLocatedTweet)-
    [:IS_WRITEN_FROM]->(l:Location)
WHERE l.name IN ["Liverpool","Seattle","Oxford"]
WITH count(et) AS englishReplies

MATCH (st:Tweet)-[:IS_A_REPLY_OF]->(r:GeoLocatedTweet)-
    [:IS_WRITEN_FROM]->(l:Location)
WHERE l.name IN ["Madrid","Valencia","Sevilla"]
WITH count(st) AS spanishReplies, englishReplies AS englishReplies

RETURN (englishReplies)*1.0/spanishReplies AS ratio;
+-----+
| ratio |
+-----+
| 0.18461538461538463 |
+-----+

1 row available after 3996 ms, consumed after another 6 ms
neo4j>

```

Ejercicio 3: MongoDB (20 %)

Para este ejercicio considera la base de datos descrita en el documento “*Diseño de una base de datos para una app de mensajería instantánea*” que se encuentra en los materiales del curso.

También se necesitará la máquina virtual LinuxMint que contiene una instalación de MongoDB y la base de datos ya cargada. Para este ejercicio no es necesario cargar ningún dato adicional.

Inicia el servicio de MongoDB y accede a los datos con los siguientes comandos:

```
sudo systemctl start mongod
```

Deja pasar unos segundos para que el servicio arranque y después ejecuta:

```
mongo
```

Y finalmente deberás entrar en la base de datos en cuestión:

```
use mensajeria
```

Para más información, lee el manual de la máquina virtual que encontrarás en los recursos de la asignatura.

Se pide proporcionar las sentencias (texto) para el shell de MongoDB y los resultados que se obtienen (haciendo una captura de pantalla) para las siguientes operaciones:

Consulta 1 (15%)

De la colección *Usuarios_Individuales* hacer un recuento de los usuarios que tengan entre 30 y 35 años de edad (ambos inclusive). Se requiere que la consulta solamente devuelva el recuento de usuarios.

Consulta 2 (15%)

De la colección *Grupos_usuarios* contar aquellos grupos de los que Ismael es miembro. Se requiere que la query solamente devuelva el recuento de grupos.

Consulta 3 (20%)

De la colección *Grupos_usuarios* mostrar el campo "Identificador" (no el campo "_id") y los nombres de los miembros de aquellos grupos de los que Ismael es miembro.

Consulta 4 (25%)

De la colección *Usuarios_bloqueados* listar los 15 usuarios con más bloqueos. Se requiere que la query devuelva el nombre de usuario y el recuento de bloqueos. Podemos asumir que no hay nombres repetidos.

Consulta 5 (25%)

De la colección *Usuarios_bloqueados* listar los nombres de los usuarios bloqueadores de la ciudad de Barcelona y el número total de bloqueos (o el recuento de las veces que aparecen en la lista Bloqueos de cada documento).

Propuesta de solución

Consulta 1:

```
> db.Usuarios_Individuales.find({Edad:{$gte:30, $lte:35}}).count();
11
```

Consulta 2:

```
> db.Grupos_usuarios.find({"Miembros.Nombre":"Ismael"}).count();
7
```

Consulta 3:

```
> db.Grupos_usuarios.find({"Miembros.Nombre":"Ismael"},
{"Identificador":1, "Miembros.Nombre":1, "_id":0});
{ "Identificador" : "G-3455", "Miembros" : [ { "Nombre" : "Ismael"
}, { "Nombre" : "Alberto" }, { "Nombre" : "Pedro" }, { "Nombre" :
"Teodoro" }, { "Nombre" : "Sandra" } ] }
{ "Identificador" : "G-3466", "Miembros" : [ { "Nombre" : "Ismael"
}, { "Nombre" : "Alberto" }, { "Nombre" : "Pedro" }, { "Nombre" :
"Teodoro" }, { "Nombre" : "Sandra" } ] }
{ "Identificador" : "G-3475", "Miembros" : [ { "Nombre" : "Javier"
}, { "Nombre" : "Ramón" }, { "Nombre" : "Victor" }, { "Nombre" :
"Pilar" }, { "Nombre" : "Ismael" } ] }
{ "Identificador" : "G-3476", "Miembros" : [ { "Nombre" : "Antonio"
}, { "Nombre" : "Laura" }, { "Nombre" : "Ismael" }, { "Nombre" :
"Javier" }, { "Nombre" : "Victor" } ] }
{ "Identificador" : "G-3484", "Miembros" : [ { "Nombre" : "Ismael"
}, { "Nombre" : "Alberto" }, { "Nombre" : "Pedro" }, { "Nombre" :
"Teodoro" }, { "Nombre" : "Sandra" } ] }
{ "Identificador" : "G-3488", "Miembros" : [ { "Nombre" : "Ismael"
}, { "Nombre" : "Alberto" }, { "Nombre" : "Pedro" }, { "Nombre" :
"Teodoro" }, { "Nombre" : "Sandra" } ] }
{ "Identificador" : "G-3494", "Miembros" : [ { "Nombre" : "Ismael"
}, { "Nombre" : "Alberto" }, { "Nombre" : "Pedro" }, { "Nombre" :
"Teodoro" }, { "Nombre" : "Sandra" } ] }
>
```

Consulta 4:

```
db.Usuarios_bloqueados.aggregate([
  { $project: { "_id":"$Usuario_bloqueado.Nombre", "TotalBloqueos"
: {$size : "$Bloqueos"} } },
  { $sort: { "TotalBloqueos": -1 } },
```



```
{ $limit: 15 }
]);
```

Nota: los usuarios que aparecen en la parte final de la lista con dos bloqueos pueden ser diferentes. Dependerá de cómo se escriba la consulta y de otros factores como la manera en la que la base de datos accede a los ficheros de datos. **Para los tres últimos usuarios se acepta como correcto cualquier usuario que tenga dos bloqueos. Como el criterio de ordenación no está especificado, los usuarios pueden tener diferente orden del que se ve en el resultado de nuestra consulta.**

```
{ "_id" : "Antonio", "TotalBloqueos" : 4 }
{ "_id" : "Lucía", "TotalBloqueos" : 4 }
{ "_id" : "Luis", "TotalBloqueos" : 4 }
{ "_id" : "Marta", "TotalBloqueos" : 4 }
{ "_id" : "Alberto", "TotalBloqueos" : 4 }
{ "_id" : "Alejandro", "TotalBloqueos" : 4 }
{ "_id" : "Cristina", "TotalBloqueos" : 4 }
{ "_id" : "Pablo", "TotalBloqueos" : 4 }
{ "_id" : "Rosa", "TotalBloqueos" : 4 }
{ "_id" : "Eva", "TotalBloqueos" : 4 }
{ "_id" : "Teodoro", "TotalBloqueos" : 3 }
{ "_id" : "Mariano", "TotalBloqueos" : 3 }
{ "_id" : "Victor", "TotalBloqueos" : 2 }
{ "_id" : "Alfredo", "TotalBloqueos" : 2 }
{ "_id" : "Gabriel", "TotalBloqueos" : 2 }
>
```

Consulta 5:

```
db.Usuarios_bloqueados.aggregate([
  { $unwind: "$Bloqueos" },
  { $match: { "Bloqueos.Usuario_bloqueador.Ciudad" : "Barcelona" } },
  { $project: { "_id":0,
    "Nombre":"$Bloqueos.Usuario_bloqueador.Nombre" } },
  { $group: { "_id":"$Nombre", "TotalBloqueos":{$sum:1} } },
  { $sort: { "TotalBloqueos": -1 } },
]);
```

Retorna:

```
{ "_id" : "Teodoro", "TotalBloqueos" : 2 }
{ "_id" : "Alberto", "TotalBloqueos" : 2 }
{ "_id" : "Cristina", "TotalBloqueos" : 2 }
{ "_id" : "Eva", "TotalBloqueos" : 2 }
```

Ejercicio 4: MongoDB (20 %)

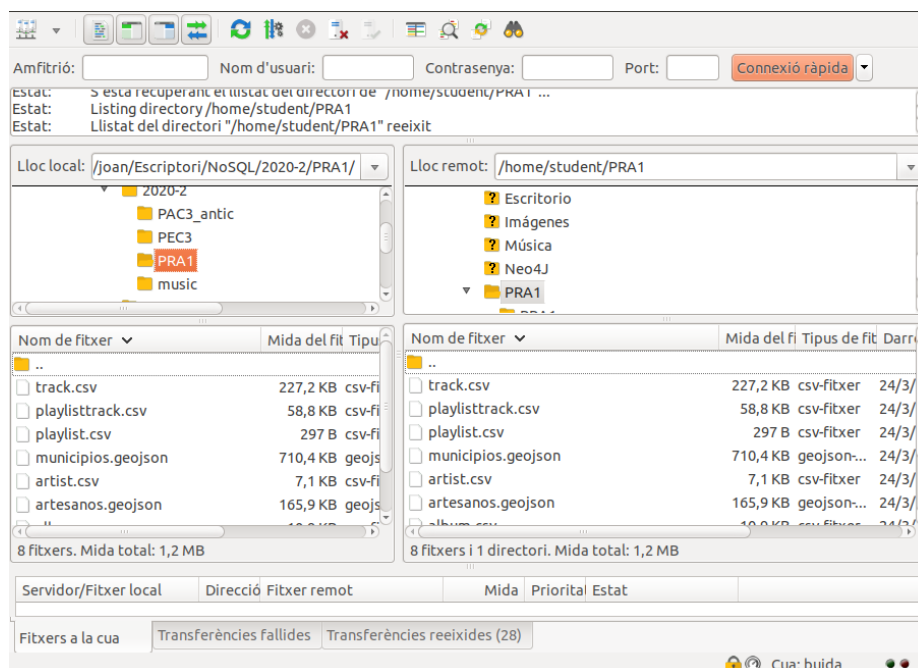
Para la realización de este ejercicio se necesitará la “*Máquina virtual LinuxMint*” que contiene una instalación de MongoDB y la información contenida en el fichero suministrado con nombre **PRA1.zip**. Al descomprimir el fichero, se observa que éste incorpora el siguiente fichero de datos en formato GeoJSON dentro del directorio *PRA1*:

- *albergues.geojson*: contiene información de las localizaciones de un conjunto de albergues de la comunidad de La Rioja.
- *artesanos.geojson*: contiene información de las localizaciones de un conjunto de artesanos de la comunidad de La Rioja.
- *municipios.geojson*: contiene información de las áreas correspondientes a los municipios de la comunidad de La Rioja.

Apartado A (25%): Realizar las siguientes acciones con el fin de poder incorporar la información suministrada en el servidor MongoDB:

1. Mediante SFTP se debe transferir a la máquina virtual los ficheros de datos en formato *geojson* mencionados. *Adjuntar captura de pantalla de la acción utilizando Filezilla.*

Propuesta de solución



2. Una vez realizada la conexión con la máquina virtual vía ssh, se debe realizar la importación de los datos de los ficheros proporcionados al servidor MongoDB. La base de datos de destino, cuyo nombre será `pra1`, incluirá la colección `albergues` donde se almacenará la información suministrada en el fichero `albergues.geojson`, una colección `municipios` donde se almacenará la información suministrada en el fichero `municipios.geojson` y una colección `artesanos` donde se almacenará la información suministrada en el fichero `artesanos.geojson`. Se puede observar que el fichero contiene información de más de un documento en formato JSON. Cada uno de estos elementos constituirá un documento distinto en la colección. *Adjuntar captura de pantalla de la ejecución de la sentencia de importación.*

Propuesta de solución

```
ssh student@localhost -p 2222
```

```
mongoimport --db pra1 --collection albergues < PRA1/albergues.geojson
```

```
mongoimport --db pra1 --collection municipios < PRA1/municipios.geojson
```

```
mongoimport --db pra1 --collection artesanos < PRA1/artesanos.geojson
```

```
student@BBDDNoSQL-VM:~$ mongoimport --db pra1 --collection albergues < PRA1/albergues.geojson
2021-03-26T16:45:22.598+0100    connected to: localhost
2021-03-26T16:45:22.687+0100    imported 12 documents
student@BBDDNoSQL-VM:~$
student@BBDDNoSQL-VM:~$ mongoimport --db pra1 --collection municipios < PRA1/municipios.geojson
2021-03-26T16:45:22.717+0100    connected to: localhost
2021-03-26T16:45:22.786+0100    imported 175 documents
student@BBDDNoSQL-VM:~$
student@BBDDNoSQL-VM:~$ mongoimport --db pra1 --collection artesanos < PRA1/artesanos.geojson
2021-03-26T16:45:22.794+0100    connected to: localhost
2021-03-26T16:45:22.822+0100    imported 130 documents
student@BBDDNoSQL-VM:~$
```

3. Una vez realizada la importación de los datos en la base de datos `pra1`, mediante Robo3T, visualizar la estructura de los agregados que componen un documento de la colección. *Adjuntar captura de pantalla de cada resultado.*

Robo 3T - 1.4

File View Options Window Help

New release available. Find out [what's new in Robo 3T](#) - [Download here](#).

MongoServer (6)

- System
- config
- films
- mensajería
- pra1
 - Collections (3)
 - albergues
 - artesanos
 - municipios
 - Functions
 - Users

db.getCollection('albergues').find({})

albergues 0.002 sec.

Key	Value	Type
(1) ObjectId("605388e997f7b3d2d83...")	{ 3 fields }	Object
_id	ObjectId("605388e997f7b3d2d8364179")	ObjectId
geometry	{ 2 fields }	Object
type	Point	String
coordinates	[3 elements]	Array
[0]	-2.296772	Double
[1]	42.18808	Double
[2]	0	Int32
properties	{ 13 fields }	Object
T238_ID	1509083	String
T238_NOMBRE	ALBERGUE JUVENIL HAYEDO DE SANTIAGO	String
T238_CAPA_SIMPLE	T238	String
T238_000_INEMUNICIPIO	26098	String
T238_000_NUCL_URB	26098000101	String
T238_000_DIR_VIALES	CIPRIANO MARTÍNEZ, CALLE	String
T238_000_DIR_VIALES_DENO	Calle Cipriano Martínez	String
T238_000_DIR_NUM	29	Int32
T238_000_DIR_COM	null	Null
T238_000_TELEFONO	900200272	String
T238_000_TELEFONO2	941205545	Int32
T238_000_CORREO_E	albergues@calleactiva.es	String
T238_000_URL	http://www.calleactiva.es/albergues	String
(2) ObjectId("605388e997f7b3d2d83...")	{ 3 fields }	Object

Logs

Robo 3T - 1.4

File View Options Window Help

New release available. Find out [what's new in Robo 3T](#) - [Download here](#).

MongoServer (6)

- System
- config
- films
- mensajeria
- pra1
 - Collections (3)
 - albergues
 - artesanos**
 - municipios
 - Functions
 - Users

db.getCollection('alber... x db.getCollection('artesa... x

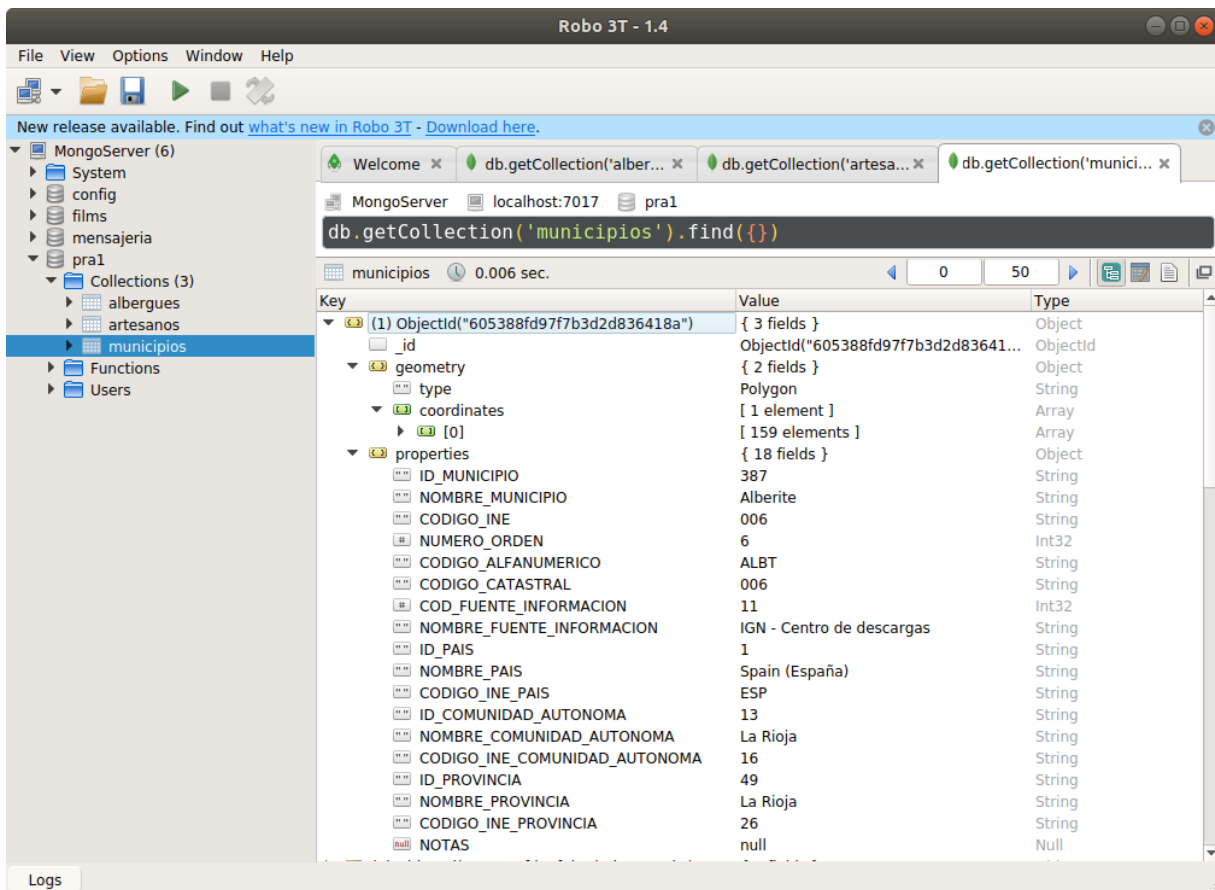
MongoServer localhost:7017 pra1

db.getCollection('artesanos').find({})

artesanos 0.003 sec.

Key	Value	Type
(1) ObjectId("6053891797f7b3d2d8...")	{ 3 fields }	Object
_id	ObjectId("6053891797f7b3d2d836614d")	ObjectId
geometry	{ 2 fields }	Object
type	Point	String
coordinates	[3 elements]	Array
[0]	-2.540576	Double
[1]	42.435187	Double
[2]	0	Int32
properties	{ 25 fields }	Object
T214_ID	1491342	String
T214_NOMBRE	José Manuel Fajardo Lozano	String
T214_CAPA_SIMPLE	T214	String
T214_CAPA_SIMPLE_DENO	Artesanos	String
T214_214_NUMREG	264	String
T214_214_CATEG	2	String
T214_214_CATEG_DENO	Artesanía de bienes de consumo	String
T214_214_OFICIOS	Alfarero	String
T214_214_ESTADO	AA	String
T214_214_ESTADO_DENO	Artesano individual	String
T214_214_NOMCOM	Alfarería Hermanos Fajardo Lozano	String
T214_000_DIR_VIALES	LOGROÑO (DE), CARRETERA	String
T214_000_DIR_VIALES_DENO	Carretera de Logroño	String
T214_000_DIR_NUM	null	Null
T214_000_DIR_COM	km 9	String
T214_000_CODIGO_POSTAL	26370	String
T214_000_NUCL_URB	26105000101	String
T214_000_INEMUNICIPIO	26105	String
T214_000_TELEFONO	941440476	String
T214_000_CORREO_E	alfareriahermanosfajardofl@hotmail.com	String
T214_000_WEB_LP	null	Null
T214_214_ACTIVIDAD	Elaboración de productos de alfarería, botijos, cántaros,...	String
T214_000_OBSERVAC	null	Null
T214_000_THUMBNAIL	null	Null
IMAGENES	<a target='_blank' href='https://ias1.larioja.org/iderioja...'	String

Logs



4. Debido a que cada documento de la colección *albergues* y *artesanos* contienen un elemento *geometry* en formato GeoJSON, se debe crear el correspondiente índice para poder realizar las debidas consultas utilizando este elemento. *Adjuntar captura de pantalla de la ejecución de la sentencia de creación.*

Propuesta de solución

```
mongo pra1
```

```
db.artesanos.createIndex( { geometry : "2dsphere" } )
```

```
db.albergues.createIndex( { geometry : "2dsphere" } )
```

```
> db.artesanos.createIndex( { geometry : "2dsphere" } )
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
> db.albergues.createIndex( { geometry : "2dsphere" } )
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
> 
```

Apartado B (25%): Se pide mostrar el nombre, el oficio y el teléfono de aquellos artesanos que estén ubicados dentro del área correspondiente al municipio cuyo nombre es “*Torrecilla en Cameros*”. Argumentar la consulta y adjuntar captura de pantalla que incluya sentencia de ejecución y resultado

Propuesta de solución:

Para ello, necesitamos obtener la ubicación de “*Torrecilla en Cameros*” (Polygon) que almacenamos en la variable `mun` para posteriormente usar en la consulta el comando `$geoWithin` para indicar que la localización del artesano esté dentro del área del municipio.

```
var mun = db.municipios.findOne(
{"properties.NOMBRE_MUNICIPIO" : "Torrecilla en Cameros"});

db.artesanos.find(
{"geometry": {
$geoWithin : { $geometry : mun.geometry} }
},
{ "properties.T214_NOMBRE" : 1,
"properties.T214_214_OFICIOS" : 1 ,
"properties.T214_000_TELEFONO" : 1 ,
"_id": 0
});
```

```
> var mun = db.municipios.findOne(
... {"properties.NOMBRE_MUNICIPIO" : "Torrecilla en Cameros"});
>
> db.artesanos.find(
... {"geometry": {
... $geoWithin : { $geometry : mun.geometry } }
... },
... { "properties.T214_NOMBRE" : 1,
... "properties.T214_214_OFICIOS" : 1 ,
... "properties.T214_000_TELEFONO" : 1 ,
... "_id": 0
... });
{ "properties" : { "T214_NOMBRE" : "Santiago Muro González", "T214_214_OFICIOS" : "Chacinerero-charcutero
(panceta curada) y elaborador de jamón", "T214_000_TELEFONO" : "941460209 / 941460190" } }
{ "properties" : { "T214_NOMBRE" : "Cárnicas Cameranas, S.L.", "T214_214_OFICIOS" : "Chacineros-charcut
eros (chorizo, salchichón, costillas, panceta) y elaboradores de jamón", "T214_000_TELEFONO" : "9414600
54-609435691" } }
```

Apartado C (25%): Realizar la correspondiente consulta con el fin de conocer el nombre de los diferentes municipios que se sobrevolarían si un dron volara en línea recta desde el albergue cuyo nombre es ALBERGUE MUNICIPAL ERMITA DEL CARRASQUEDO GRAÑON, con coordenadas [-3.023108, 42.437486, 0], hasta el albergue ALBERGUE JUVENIL RIO ALHAMA ALFARO, con coordenadas [-1.748228, 42.182035, 0]

Definir la ruta como una línea , argumentar las operaciones, implementarlas y mostrar los resultados obtenidos mediante una captura de pantalla.

Propuesta de solución:

La línea recta correspondiente a la ruta realizada por el supuesto vuelo se puede representar como un elemento GeoJSON de tipo *LineString* a partir de los dos puntos propuestos y que asignamos a la variable *route*.

```
var route= {
  "type": "LineString",
  "coordinates": [ [ -3.023108, 42.437486, 0 ] ,
                  [ -1.748228, 42.182035, 0 ]
                ]
}
```

Posteriormente buscamos todos los municipios cuyas fronteras tienen coordenadas que se intersectan con la ruta especificada . Esto se hace con la siguiente consulta:

```
db.municipios.find(
{ "geometry" : { $geoIntersects : { $geometry : route } } },
{ "properties.NOMBRE_MUNICIPIO" : 1 , "_id" : 0 }
);
```

Obteniendo el siguiente resultado:


```

..var route= {"type": "LineString","coordinates": [ [ -3.023108, 42.437486,0 ] , [-1.748228, 42.182035, 0 ]
] ] }
> db.municipios.find(
... { "geometry" : { $geoIntersects : { $geometry : route } } },
... { "properties.NOMBRE_MUNICIPIO" : 1 , "_id" : 0 }
... );
{"properties" : { "NOMBRE_MUNICIPIO" : "Sotés" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Sorzano" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Santa Coloma" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Santo Domingo de la Calzada" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Sojuela" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Grañón" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Rincón de Soto" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Leza de Río Leza" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Robres del Castillo" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Mancomunidad de Nalda, Sorzano y Viguera" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Nalda" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Bergasillas Bajera" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Cañas" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Arnedo" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Arenzana de Abajo" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Alesanco" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Aldeanueva de Ebro" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Alfaro" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Autol" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Canillas de Río Tuerto" } }
Type "it" for more
> it
{"properties" : { "NOMBRE_MUNICIPIO" : "Arenzana de Arriba" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Bergasa" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Nájera" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Corporales" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Cordovin" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Cirueña" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Quel" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Torrecilla sobre Alesanco" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Clavijo" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Lagunilla del Jubera" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Santa Engracia del Jubera" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Bezares" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Daroca de Rioja" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Soto en Cameros" } }
{"properties" : { "NOMBRE_MUNICIPIO" : "Ocón" } }
>

```

Apartado D (25%): Dada la consulta siguiente:

```

db.artesanos.find(
  { "properties.T214_214_ACTIVIDAD": { $regex: /conservas/ } },
  { _id:0,"properties.T214_NOMBRE":1 }
)

```

Indicar qué índices crearías para mejorar el rendimiento de este tipo de consultas. Analiza el plan de ejecución de las consultas, antes y después de la creación del índice

Propuesta de solución:

Utilizado el método `explain("executionStats")` MongoDB ejecuta el optimizador de consultas para elegir el plan de ejecución óptimo, ejecuta el plan elegido hasta su finalización y devuelve estadísticas que describen la ejecución del plan óptimo. Lo primero es ejecutar la consulta y ver el plan de ejecución. Eso se consigue con:

```
db.artesanos.find(
  { "properties.T214_214_ACTIVIDAD": { $regex: /conservas/ } },
  { _id:0,"properties.T214_NOMBRE":1 }
).explain("executionStats")
```

```
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    "namespace" : "pral.artesanos",
    "indexFilterSet" : false,
    "parsedQuery" : {
      "properties.T214_214_ACTIVIDAD" : {
        "$regex" : "conservas"
      }
    },
    "winningPlan" : {
      "stage" : "PROJECTION",
      "transformBy" : {
        "_id" : 0,
        "properties.T214_NOMBRE" : 1
      },
      "inputStage" : {
        "stage" : "COLLSCAN",
        "filter" : {
          "properties.T214_214_ACTIVIDAD" : {
            "$regex" : "conservas"
          }
        }
      },
      "direction" : "forward"
    },
    "rejectedPlans" : [ ]
  },
  "executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 6,
    "executionTimeMillis" : 0,
    "totalKeysExamined" : 0,
    "totalDocsExamined" : 130,
    "executionStages" : {
      "stage" : "PROJECTION",
      "nReturned" : 6,
      "executionTimeMillisEstimate" : 0,
      "works" : 132,
      "advanced" : 6,
      "needTime" : 125,
      "needYield" : 0,
      "saveState" : 1,
```

```

        "restoreState" : 1,
        "isEOF" : 1,
        "invalidates" : 0,
        "transformBy" : {
            "_id" : 0,
            "properties.T214_NOMBRE" : 1
        },
        "inputStage" : {
            "stage" : "COLLSCAN",
            "filter" : {
                "properties.T214_214_ACTIVIDAD" : {
                    "$regex" : "conservas"
                }
            },
            "nReturned" : 6,
            "executionTimeMillisEstimate" : 0,
            "works" : 132,
            "advanced" : 6,
            "needTime" : 125,
            "needYield" : 0,
            "saveState" : 1,
            "restoreState" : 1,
            "isEOF" : 1,
            "invalidates" : 0,
            "direction" : "forward",
            "docsExamined" : 130
        }
    },
    "serverInfo" : {
        "host" : "BDDNoSQL-VM",
        "port" : 27017,
        "version" : "3.6.3",
        "gitVersion" : "9586e557d54ef70f9ca4b43c26892cd55257e1a5"
    },
    "ok" : 1
}

```

A partir de la información aportada por el *queryPlanner* y *executionStats* se puede observar un alto número de trabajos, works, (132) debido al uso de la etapa *COLLSCAN*, escaneo y filtrado del valor del campo `properties.T214_214_ACTIVIDAD` de todos los documentos que componen la colección, 130. Posteriormente se ha realizado una operación de proyección, *PROJECTION*, con el fin de mostrar el valor del campo `properties.T214_NOMBRE`.

Con el fin de evitar este alto número de trabajos se debe optimizar la consulta utilizando un índice de texto sobre el campo `properties.T214_214_ACTIVIDAD` de la colección `artesanos`. Tan sólo se puede crear un índice de texto por colección. La sentencia de creación del índice sobre el campo que se utiliza en la operación de selección será la siguiente:

```
db.artesanos.createIndex({ 'properties.T214_214_ACTIVIDAD':'text'},
    { 'name': 'para consulta artesanos x actividad' });
```

```
> db.artesanos.createIndex({ 'properties.T214_214_ACTIVIDAD':'text'},
...   { 'name': 'para consulta artesanos x actividad' });
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 2,
  "numIndexesAfter" : 3,
  "ok" : 1
}
>
```

Ahora será necesario modificar la consulta, pues realizaremos la búsqueda directamente sobre el índice de texto. La consulta a utilizar será:

```
db.artesanos.find( { $text: { $search: 'conservas' } },
{ _id:0,"properties.T214_NOMBRE":1 }
)
```

Por lo que el plan de ejecución se visualizará mediante:

```
db.artesanos.find( { $text: { $search: 'conservas' } },
{ _id:0,"properties.T214_NOMBRE":1 }
).explain("executionStats")
```

```
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    "namespace" : "pral.artesanos",
    "indexFilterSet" : false,
    "parsedQuery" : {
      "$text" : {
        "$search" : "conservas",
        "$language" : "english",
        "$caseSensitive" : false,
        "$diacriticSensitive" : false
      }
    }
  },
```

```

"winningPlan" : {
  "stage" : "PROJECTION",
  "transformBy" : {
    "_id" : 0,
    "properties.T214_NOMBRE" : 1
  },
  "inputStage" : {
    "stage" : "TEXT",
    "indexPrefix" : {  },
    "indexName" : "para consulta artesanos x actividad",
    "parsedTextQuery" : {
      "terms" : ["conserva" ],
      "negatedTerms" : [ ],
      "phrases" : [ ],
      "negatedPhrases" : [ ]
    },
    "textIndexVersion" : 3,
    "inputStage" : {
      "stage" : "TEXT_MATCH",
      "inputStage" : {
        "stage" : "FETCH",
        "inputStage" : {
          "stage" : "OR",
          "inputStage" : {
            "stage" : "IXSCAN",
            "keyPattern" : {
              "_fts" : "text",
              "_ftsx" : 1
            },
            "indexName" : "para consulta artesanos x actividad",
            "isMultiKey" : true,
            "isUnique" : false,
            "isSparse" : false,
            "isPartial" : false,
            "indexVersion" : 2,
            "direction" : "backward",
            "indexBounds" : {

              }
            }
          }
        }
      }
    },
    "rejectedPlans" : [ ]
  },
  "executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 6,
    "executionTimeMillis" : 0,
    "totalKeysExamined" : 6,
    "totalDocsExamined" : 6,
    "executionStages" : {

```

```

"stage" : "PROJECTION",
"nReturned" : 6,
"executionTimeMillisEstimate" : 0,
"works" : 7,
"advanced" : 6,
"needTime" : 0,
"needYield" : 0,
"saveState" : 0,
"restoreState" : 0,
"isEOF" : 1,
"invalidates" : 0,
"transformBy" : {
  "_id" : 0,
  "properties.T214_NOMBRE" : 1
},
"inputStage" : {
  "stage" : "TEXT",
  "nReturned" : 6,
  "executionTimeMillisEstimate" : 0,
  "works" : 7,
  "advanced" : 6,
  "needTime" : 0,
  "needYield" : 0,
  "saveState" : 0,
  "restoreState" : 0,
  "isEOF" : 1,
  "invalidates" : 0,
  "indexPrefix" : { },
  "indexName" : "para consulta artesanos x actividad",
  "parsedTextQuery" : {
    "terms" : ["conserva" ],
    "negatedTerms" : [ ],
    "phrases" : [ ],
    "negatedPhrases" : [ ]
  },
  "textIndexVersion" : 3,
  "inputStage" : {
    "stage" : "TEXT_MATCH",
    "nReturned" : 6,
    "executionTimeMillisEstimate" : 0,
    "works" : 7,
    "advanced" : 6,
    "needTime" : 0,
    "needYield" : 0,
    "saveState" : 0,
    "restoreState" : 0,
    "isEOF" : 1,
    "invalidates" : 0,
    "docsRejected" : 0,
    "inputStage" : {
      "stage" : "FETCH",
      "nReturned" : 6,
      "executionTimeMillisEstimate" : 0,
      "works" : 7,

```

```

"advanced" : 6,
"needTime" : 0,
"needYield" : 0,
"saveState" : 0,
"restoreState" : 0,
"isEOF" : 1,
"invalidates" : 0,
"docsExamined" : 6,
"alreadyHasObj" : 0,
"inputStage" : {
  "stage" : "OR",
  "nReturned" : 6,
  "executionTimeMillisEstimate" : 0,
  "works" : 7,
  "advanced" : 6,
  "needTime" : 0,
  "needYield" : 0,
  "saveState" : 0,
  "restoreState" : 0,
  "isEOF" : 1,
  "invalidates" : 0,
  "dupsTested" : 6,
  "dupsDropped" : 0,
  "recordIdsForgotten" : 0,
  "inputStage" : {
    "stage" : "IXSCAN",
    "nReturned" : 6,
    "executionTimeMillisEstimate" : 0,
    "works" : 7,
    "advanced" : 6,
    "needTime" : 0,
    "needYield" : 0,
    "saveState" : 0,
    "restoreState" : 0,
    "isEOF" : 1,
    "invalidates" : 0,
    "keyPattern" : {
      "_fts" : "text",
      "_ftsx" : 1
    },
    "indexName" : "para consulta artesanos x actividad",
    "isMultiKey" : true,
    "isUnique" : false,
    "isSparse" : false,
    "isPartial" : false,
    "indexVersion" : 2,
    "direction" : "backward",
    "indexBounds" : {

    },
    "keysExamined" : 6,
    "seeks" : 1,
    "dupsTested" : 6,
    "dupsDropped" : 0,

```

```

        "seenInvalidated" : 0
    }
}
}
}
}
},
"serverInfo" : {
    "host" : "BBDDNoSQL-VM",
    "port" : 27017,
    "version" : "3.6.3",
    "gitVersion" : "9586e557d54ef70f9ca4b43c26892cd55257e1a5"
},
"ok" : 1
}

```

A partir de la información aportada por el *queryPlanner* se puede observar que se ha realizado un acceso directo por valor, *FETCH*, a partir del resultado IXSCAN realizado sobre el índice cuyo nombre es `para consulta artesanos x actividad` y una operación de proyección, *PROJECTION*, con el fin de mostrar el valor del campo `properties.T214_NOMBRE`.

La existencia del índice evita que el *queryPlanner* utilice la operación COLLSCAN que se realizaba antes de la creación del índice.

A partir de la información aportada por las estadísticas de ejecución, *executionStats*, se puede observar con más detalle que el número de trabajos, *works*, ha disminuido notablemente con el uso del índice: 7.

Criterios de valoración

En cada ejercicio se valorará la validez de la solución y la claridad de la argumentación. Cada ejercicio tiene indicado en el enunciado su peso en la valoración final.

Formato y fecha de entrega

Tenéis que enviar la PRA1 al buzón de Entrega y registro de EC disponible en el aula (apartado Evaluación). El formato del archivo que contiene vuestra solución puede ser .pdf, .odt, .doc y .docx. Para otras opciones, por favor, contactar previamente con vuestro profesor colaborador. El nombre del fichero debe contener el código de la asignatura, vuestro apellido y vuestro nombre, así como el número de actividad (PRA1). Por ejemplo apellido1_nombre_nosql_pra1.pdf

La fecha límite para entregar la PRA1 es el **6 de junio de 2021**.

Propiedad intelectual

Al presentar una práctica o PEC que haga uso de recursos ajenos, se tiene que presentar junto con ella un documento en que se detallen todos ellos, especificando el nombre de cada recurso, su autor, el lugar donde se obtuvo y su estatus legal: si la obra está protegida por el copyright o se acoge a alguna otra licencia de uso (Creative Commons, licencia GNU, GPL etc.). El estudiante tendrá que asegurarse que la licencia que sea no impide específicamente su uso en el marco de la práctica o PEC. En caso de no encontrar la información correspondiente tendrá que asumir que la obra está protegida por el copyright.

Será necesario, además, adjuntar los ficheros originales cuando las obras utilizadas sean digitales, y su código fuente, si así corresponde.