Uso de bases de datos NoSQL
PRA1

Javier Gómez de Diego

Instrucciones

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

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.

Las respuestas se mostrarán con una línea verde en el margen izquierdo para distinguirlas fácilmente del enunciado.

Ejercicio 1: Cassandra (30%)

Al tener un volumen de datos creciente, para el análisis de la información y para mejorar el rendimiento de algunas aplicaciones, la empresa se plantea utilizar una base de datos que implemente un modelo agrupado como Cassandra.

Tomaremos como referencia los agregados de la PAC 1. Os recomendamos que reviséis la estructura general en la propuesta de solución para tener un mejor contexto. Los datos los tendréis que cargar vosotros siguiendo las instrucciones detalladas a continuación.

Carga de datos

Se tendrán que ejecutar los siguientes comandos CQL para crear y rellenar las tablas o familias de columnas.

Las instrucciones siguientes parten de una conexión a la línea de comandos de la base de datos y con el servicio de Cassandra ya arrancado. Partiremos del siguiente símbolo del sistema (*command prompt*) en la línea de comandos (para llegar a este punto podéis seguir el manual "Uso de máquina virtual - Bases de datos NoSQL"):

```
cqlsh>
```

Primero creamos un keyspace llamado sport_app con el siguiente comando:

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

Indicamos que queremos utilizar el keyspace creado ejecutando:

```
use sport_app;
```

Que nos cambiará el *command prompt* a:

```
cqlsh:sport_app>
```

Una vez estamos en el keyspace correcto, creamos la primera agrupación `month_category_runner` ejecutando el archivo `01_drop_and_create_month_category_runner.cql` que encontraréis junto al enunciado. Las instrucciones para ejecutar archivos de comandos las encontraréis en los vídeos del principio de la PAC. Tened en cuenta que **este archivo borra y crea de nuevo la tabla o agrupación `month_category_runner`, por lo que se borrarán todos los datos y cambios que hayáis podido realizar**. Si queréis guardar los cambios hechos en esta tabla, podéis cambiarle el nombre y de esta manera el script no la borrará.

El siguiente paso es rellenar los datos en la tabla. Para ello, tenéis que ejecutar el archivo `02_insert_month_category_runner.cql` igual que se ha ejecutado el archivo anterior. Si la tabla ya estaba rellenada anteriormente puede que la ejecución de este script modifique algunos datos. Tenedlo en cuenta si ya habéis realizado algunos cambios.

Para comprobar que las filas se han insertado correctamente podemos ejecutar:

```
SELECT     month, category, runner_id, club, race_name, race_type,
           TODATE(race_date) AS race_date, net_time
FROM  month_category_runner;
```

Notar que para dar un formato más compacto en la tabla se han utilizado las funciones TODATE para la columna `race_date`.

La consulta retorna:

```
 month  | category | runner_id    | club             | race_name                | race_type      | race_date  | net_time
--------+----------+--------------+------------------+--------------------------+----------------+------------+------------------
 202111 |   SENIOR | 11.111.111-A | CORNELLA ATLETIC |   BEHOBIA-SAN SEBASTIAN | URBAN RACE 20K | 2021-11-14 | 01:29:17.000000000
 202112 |   SENIOR | 22.222.222-A |     THE RUN CLUB | 40 TROFEO AKILES-TRIMAD | URBAN RACE 10K | 2021-12-08 | 00:46:01.000000000
 202112 |   SENIOR | 22.222.222-A |     THE RUN CLUB |         MADRID MARATHON | URBAN RACE 42K | 2021-09-26 | 03:15:49.000000000
 202112 |   SENIOR | 44.444.444-A |     THE RUN CLUB | SAN SILVESTRE VALLECANA | URBAN RACE 10K | 2021-12-31 | 00:45:32.000000000
 202105 |   JUNIOR | 33.333.333-A |     CAN MERCADER |              CROSS GAVA |      CROSS 10K | 2021-05-29 | 00:37:59.000000000
 202105 |   JUNIOR | 33.333.333-A |     CAN MERCADER |        POPULAR CORNELLA | URBAN RACE 10K | 2021-05-10 | 00:35:01.000000000
 202109 |   SENIOR | 11.111.111-A | CORNELLA ATLETIC |               JEAN BOUIN | URBAN RACE 10K | 2021-09-20 | 00:41:47.000000000
 202109 |   SENIOR | 11.111.111-A | CORNELLA ATLETIC |                 LA MERCE | URBAN RACE 10K | 2021-09-01 | 00:39:47.000000000

(8 rows)
cqlsh:sport_app>
```

Este es el formato recomendado para adjuntar los resultados de Cassandra en vuestras respuestas a los ejercicios. La fuente Courier alinea mejor los resultados ya que todas las letras y símbolos tienen el mismo ancho. Si la hacéis lo suficientemente pequeña los datos caben perfectamente sin saltos de línea entre medias. Las consultas que se solicitan no tienen tantas columnas como la anterior.

A continuación, creamos la segunda agrupación de columnas llamada club_competitions_category. Para crear la tabla tendréis que ejecutar el archivo 03_drop_and_create_club_competitions_category.cql y para cargar los datos el archivo 04_insert_club_competitions_category.cql. Si seleccionamos los registros de esta tabla con la siguiente consulta nos retornará:

```
SELECT club, race_name, TODATE(race_date) as race_date,
            category, runner_id, race_type, net_time
FROM  club_competitions_category;
```

Que retorna:

```
 club             | race_name                | race_date  | category | runner_id    | race_type      | net_time
------------------+--------------------------+------------+----------+--------------+----------------+------------------
 CORNELLA ATLETIC |   BEHOBIA-SAN SEBASTIAN | 2021-11-13 |   SENIOR | 11.111.111-A | URBAN RACE 20K | 01:29:17.000000000
 CORNELLA ATLETIC |               JEAN BOUIN | 2021-09-19 |   SENIOR | 11.111.111-A | URBAN RACE 10K | 00:41:47.000000000
 CORNELLA ATLETIC |                 LA MERCE | 2021-08-31 |   SENIOR | 11.111.111-A | URBAN RACE 10K | 00:39:47.000000000
     THE RUN CLUB | 40 TROFEO AKILES-TRIMAD | 2021-12-07 |   SENIOR | 22.222.222-A | URBAN RACE 10K | 00:46:01.000000000
     THE RUN CLUB |         MADRID MARATHON | 2021-09-25 |   SENIOR | 22.222.222-A | URBAN RACE 42K | 03:15:49.000000000
     THE RUN CLUB | SAN SILVESTRE VALLECANA | 2021-12-30 |   SENIOR | 44.444.444-A | URBAN RACE 10K | 00:45:32.000000000
     CAN MERCADER |              CROSS GAVA | 2021-05-28 |   JUNIOR | 33.333.333-A |      CROSS 10K | 00:37:59.000000000
     CAN MERCADER |        POPULAR CORNELLA | 2021-05-09 |   JUNIOR | 33.333.333-A | URBAN RACE 10K | 00:35:01.000000000

(8 rows)
cqlsh:sport_app>
```

Consultas Cassandra

Con los datos que habéis cargado en el apartado anterior, se requiere realizar las consultas detalladas y adjuntar su resultado en modo texto (no capturas de pantalla).

Puede que algunas consultas retornen error y no se puedan ejecutar directamente. En tal caso, debéis adjuntar las consultas que os dan error y los comandos o acciones 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 pide en el enunciado,** especialmente para aquellas consultas que no son directas. También se tendrá en cuenta que la solución sea la más óptima posible.

En algunos casos puede que tengáis que obtener datos adicionales para ejecutar las consultas, puede que tengáis que crear un índice, o puede que tengáis que crear otra tabla. **Tenéis total libertad para realizar todas aquellas acciones necesarias para que la consulta retorne los datos solicitados en el enunciado, pero siempre teniendo en cuenta cuál es la opción más eficiente**. Se valorará con más puntuación el procedimiento más eficiente.

Debéis considerar que esta base de datos en un entorno real tendrá mucho volumen de datos. Por lo tanto, si se requieren consultas previas para recuperar datos necesarios para otra consulta, **no se puede consultar toda la tabla y obtener los datos necesarios del listado completo de la tabla.** Se deben consultar los datos de manera filtrada para que la consulta previa retorne la información concreta que se necesita para resolver el ejercicio. En un escenario real con muchos registros no es viable seleccionar toda una tabla entera y buscar los datos manualmente en pantalla.

Siempre se deben recuperar los datos solicitados a partir de la información facilitada en el enunciado.

No se puede utilizar la cláusula ALLOW FILTERING.

Por último, si se tiene que realizar alguna modificación de datos, se valorará que la base de datos quede en un estado consistente. Además, tendréis que adjuntar una consulta que demuestre que los datos han quedado correctamente actualizados.

Para adjuntar los comandos que vayáis ejecutando en la práctica, por favor, hacedlo en modo texto tal y como se ha realizado en el ejemplo de carga de datos. Para documentar el resultado de la ejecución de los comandos adjuntar también el texto de la salida de la consola de comandos como se ha visto en la carga de datos.

Consulta 1.1 recuento de datos (5%):

¿Cuántos corredores han corrido durante el mes de diciembre del 2021? La consulta debe retornar una sola cifra.

Primero, creamos la tabla *month_runners* con *runner_id* y *month*, siguiendo la siguiente estructura de claves.

```
CREATE TABLE month_runners (
runner_id text,
month text,
PRIMARY KEY ((month), runner_id));
```

A continuación, insertamos los datos tras exportarlos de la tabla *month_category_runner*.

```
COPY month_category_runner (runner_id, month) TO 'export.csv';

COPY month_runners(runner_id, month) FROM 'export.csv';
```

Verificamos los datos con la siguiente consulta y los comparamos con los de la tabla original.

```
SELECT runner_id, month  FROM month_category_runner;

runner_id    | month
--------------+--------
 11.111.111-A | 202111
 22.222.222-A | 202112
 22.222.222-A | 202112
 44.444.444-A | 202112
 33.333.333-A | 202105
 33.333.333-A | 202105
 11.111.111-A | 202109
 11.111.111-A | 202109

SELECT runner_id, month  FROM month_runners;

 runner_id    | month
--------------+--------
 11.111.111-A | 202111
 22.222.222-A | 202112
 44.444.444-A | 202112
 33.333.333-A | 202105
 11.111.111-A | 202109
```

Vemos que la estructura de claves permite obtener los datos deseados en la consulta (no queremos que se repitan combinaciones corredor-mes). Por lo tanto, realizamos la consulta.

```
SELECT count(*) FROM month_runners WHERE month = '202112';

 count
-------
     2
```

Consulta 1.2 recuento de datos (5%):

Queremos saber el número total de corredores que tenemos en la base de datos. La consulta debe retornar una sola cifra. El ejercicio devuelve un warning. ¿Podrías explicar por qué?

Primero, creamos la tabla *runners* con *runner_id*.

```
CREATE TABLE runners (
runner_id text,
PRIMARY KEY (runner_id));
```

A continuación, insertamos los datos tras exportarlos de la tabla *month_category_runner*.

```
COPY month_category_runner (runner_id) TO 'export_rid.csv';

COPY runners(runner_id) FROM 'export_rid.csv';
```

Verificamos los datos con la siguiente consulta.

```
SELECT * FROM runners;

runner_id
--------------
 11.111.111-A
 33.333.333-A
 22.222.222-A
 44.444.444-A
```

Ya podemos realizar la consulta.

```
SELECT count(*) FROM runners;

 count
-------
     4

Warnings :
Aggregation query used without partition key
```

El `warning` se produce porque la función **`count(*)`** agrega todas las filas de la tabla debido a que no se ha aplicado una cláusula **`WHERE`**. El `warning` nos indica que la función no es eficiente.

Consulta 1.3 listado (5%):

De los corredores del club “THE RUN CLUB”, mostrar el nombre de la carrera, la categoría, el identificador del corredor y el tiempo de carrera (net_time).

```
SELECT race_name, category, runner_id, net_time
FROM club_competitions_category
WHERE club='THE RUN CLUB';
```

```
race_name                | category | runner_id    | net_time
-------------------------+----------+--------------+--------------------
 40 TROFEO AKILES-TRIMAD |   SENIOR | 22.222.222-A | 00:46:01.000000000
        MADRID MARATHON  |   SENIOR | 22.222.222-A | 03:15:49.000000000
 SAN SILVESTRE VALLECANA |   SENIOR | 44.444.444-A | 00:45:32.000000000
```

Consulta 1.4 listado (10%)

Recuento de corredores agrupado por mes y categoría. La consulta debe retornar solamente el mes, la categoría y el recuento.

```
SELECT month, category, count(*)
FROM month_category_runner
GROUP BY month;
```

```
 month  | category | count
--------+----------+-------
 202111 |   SENIOR |     1
 202112 |   SENIOR |     3
 202105 |   JUNIOR |     2
 202109 |   SENIOR |     2
```

Consulta 1.5 listado (15%)

Listado de carreras del tipo 'URBAN RACE 10K’ en las que ha participado el club 'CORNELLA ATLETIC'. La consulta debe retornar solamente los nombres de las carreras.

Creamos un índice en la table club_competitions_category sobre race_type.

```
CREATE INDEX ON club_competitions_category (race_type);
```

Realizamos la consulta.

```
SELECT race_name
FROM club_competitions_category
WHERE club='CORNELLA ATLETIC' AND race_type='URBAN RACE 10K';
```

```
 race_name
------------
 JEAN BOUIN
 LA MERCE
```

Consulta 1.6 modificación de datos (50%)

Hay un error en la base de datos. En la carrera BEHOBIA-SAN SEBASTIAN no participaron 29000 corredores si no que lo hicieron 31000. Se solicita actualizar los valores en la base de datos. Recordad que solamente se puede elaborar una solución a partir de los datos detallados en el enunciado.

Creamos una nueva tabla temporal con race_name como PRIMARY KEY.

```
CREATE TABLE TMP_month_category_runner (month text,
category text,
runner_id text,
club text,
race_name text,
race_type text,
race_date timestamp,
total_runners int,
net_time time,
PRIMARY KEY(race_name));
```

Copiamos el contenido de la tabla month_category_runner en la temporal.

```
COPY month_category_runner(race_name, category, club, month, net_time,
race_date, race_type, runner_id, total_runners) TO 'export_tmp.csv';

COPY TMP_month_category_runner(race_name, category, club, month, net_time,
race_date, race_type, runner_id, total_runners) FROM 'export_tmp.csv';
```

Actualizamos el dato erróneo.

```
UPDATE TMP_month_category_runner
SET total_runners=31000
WHERE race_name='BEHOBIA-SAN SEBASTIAN';
```

Copiamos el contenido de la tabla temporal (actualizado) en las tablas antiguas.

```
COPY TMP_month_category_runner(month, category, runner_id, race_name,
net_time, club, race_date, race_type, total_runners) TO 'export.csv';

COPY month_category_runner(month, category, runner_id, race_name, net_time,
club, race_date, race_type, total_runners) FROM 'export.csv';

COPY TMP_month_category_runner(club, race_name, category, runner_id,
net_time, race_date, race_type, total_runners) TO 'export.csv';

COPY club_competitions_category(club, race_name, category, runner_id,
net_time, race_date, race_type, total_runners) FROM 'export.csv';
```

Comprobamos que el dato se ha actualizado correctamente.

```
SELECT race_name, total_runners FROM month_category_runner ;
```

```
 race_name               | total_runners
-------------------------+---------------
   BEHOBIA-SAN SEBASTIAN |         31000
 40 TROFEO AKILES-TRIMAD |          5000
         MADRID MARATHON |         19000
 SAN SILVESTRE VALLECANA |         30000
              CROSS GAVA |          1000
        POPULAR CORNELLA |           500
              JEAN BOUIN |         15000
                LA MERCE |         25000
```

```
SELECT race_name, total_runners FROM club_competitions_category;
```

```
 race_name               | total_runners
-------------------------+---------------
   BEHOBIA-SAN SEBASTIAN |         31000
              JEAN BOUIN |         15000
                LA MERCE |         25000
 40 TROFEO AKILES-TRIMAD |          5000
         MADRID MARATHON |         19000
 SAN SILVESTRE VALLECANA |         30000
              CROSS GAVA |          1000
        POPULAR CORNELLA |           500
```

Consulta 1.7 encuentra el error de diseño (10%)

En la familia de columnas club_competitions_category hay un error de diseño. Supongamos que tenemos que añadir datos referentes a la nueva edición de la carrera SAN SILVESTRE VALLECANA. Como es normal, muchos corredores repiten en todas las ediciones y carrera. Para constatar el error a nivel práctico, selecciona los datos de la tabla club_competitions_category (o haz un recuento de todos los registros) y después inserta el siguiente registro con la instrucción:

```
INSERT INTO club_competitions_category (club, race_name, race_date, category, runner_id, race_type, net_time, total_runners) values ('THE RUN CLUB', 'SAN SILVESTRE VALLECANA','2022-12-31', 'SENIOR', '44.444.444-A', 'URBAN RACE 10K', '00:44:29', 30500);
```

Vuelve a seleccionar la misma tabla (o haz un recuento nuevamente) y observa bien los datos que retorna la misma. Algo no ha ido como debería. ¿Qué ha fallado y por qué ha fallado?

Pista: en esta dirección web encontrarás similitudes y particularidades sobre las instrucciones insert y update en Cassandra. Tened en cuenta que en otra entrada del mismo hilo se indica PRA1 – Bases de datos NoSQL –/03/2022 pág 6 que la afirmación no es 100% precisa. Ya podemos anticipar que este no es el motivo principal de este "extraño" comportamiento, pero puede ayudar a contestar la pregunta.

Nota: no se pide ni solucionar ni implementar la solución, basta con explicar qué sucede y por qué en una extensión máxima de media página.

SELECT * FROM club_competitions_category;

club	race_name	category	runner_id	net_time	race_date	race_type	total_runners
CORNELLA ATLETIC	BEHOBIA-SAN SEBASTIAN	SENIOR	11.111.111-A	01:29:17.000000000	2021-11-14 07:30:00.000000+0000	URBAN RACE 20K	31000
CORNELLA ATLETIC	JEAN BOUIN	SENIOR	11.111.111-A	00:41:47.000000000	2021-09-20 07:00:00.000000+0000	URBAN RACE 10K	15000
CORNELLA ATLETIC	LA MERCE	SENIOR	11.111.111-A	00:39:47.000000000	2021-09-01 06:30:00.000000+0000	URBAN RACE 10K	25000
THE RUN CLUB	40 TROFEO AKILES-TRIMAD	SENIOR	22.222.222-A	00:46:01.000000000	2021-12-08 09:00:00.000000+0000	URBAN RACE 10K	5000
THE RUN CLUB	MADRID MARATHON	SENIOR	22.222.222-A	03:15:49.000000000	2021-09-26 06:45:00.000000+0000	URBAN RACE 42K	19000
THE RUN CLUB	**SAN SILVESTRE VALLECANA**	**SENIOR**	**44.444.444-A**	**00:45:32.000000000**	**2021-12-31 17:00:00.000000+0000**	**URBAN RACE 10K**	**30000**
CAN MERCADER	CROSS GAVA	JUNIOR	33.333.333-A	00:37:59.000000000	2021-05-29 06:00:00.000000+0000	CROSS 10K	1000
CAN MERCADER	POPULAR CORNELLA	JUNIOR	33.333.333-A	00:35:01.000000000	2021-05-10 08:00:00.000000+0000	URBAN RACE 10K	500

INSERT INTO club_competitions_category (club, race_name, race_date, category, runner_id, race_type, net_time, total_runners) values ('THE RUN CLUB', 'SAN SILVESTRE VALLECANA','2022-12-31', 'SENIOR', '44.444.444-A', 'URBAN RACE 10K', '00:44:29', 30500);

SELECT * FROM club_competitions_category;

club	race_name	category	runner_id	net_time	race_date	race_type	total_runners
CORNELLA ATLETIC	BEHOBIA-SAN SEBASTIAN	SENIOR	11.111.111-A	01:29:17.000000000	2021-11-14 07:30:00.000000+0000	URBAN RACE 20K	31000
CORNELLA ATLETIC	JEAN BOUIN	SENIOR	11.111.111-A	00:41:47.000000000	2021-09-20 07:00:00.000000+0000	URBAN RACE 10K	15000
CORNELLA ATLETIC	LA MERCE	SENIOR	11.111.111-A	00:39:47.000000000	2021-09-01 06:30:00.000000+0000	URBAN RACE 10K	25000
THE RUN CLUB	40 TROFEO AKILES-TRIMAD	SENIOR	22.222.222-A	00:46:01.000000000	2021-12-08 09:00:00.000000+0000	URBAN RACE 10K	5000
THE RUN CLUB	MADRID MARATHON	SENIOR	22.222.222-A	03:15:49.000000000	2021-09-26 06:45:00.000000+0000	URBAN RACE 42K	19000
THE RUN CLUB	**SAN SILVESTRE VALLECANA**	**SENIOR**	**44.444.444-A**	**00:44:29.000000000**	**2022-12-30 23:00:00.000000+0000**	**URBAN RACE 10K**	**30500**
CAN MERCADER	CROSS GAVA	JUNIOR	33.333.333-A	00:37:59.000000000	2021-05-29 06:00:00.000000+0000	CROSS 10K	1000
CAN MERCADER	POPULAR CORNELLA	JUNIOR	33.333.333-A	00:35:01.000000000	2021-05-10 08:00:00.000000+0000	URBAN RACE 10K	500

Se puede observar que el error se encuentra en que no se ha añadido una fila más, si no que la fila existente correspondiente a la edición anterior de la misma carrera **se ha actualizado/sobrescrito aparentemente**.

Como bien se explica en los enlaces incluidos en el enunciado, **`INSERT`** y **`UPDATE`** tienen efectos idénticos bajo ciertas circunstancias desde la perspectiva del usuario: al hacer un **`INSERT`** con una clave que ya existe en la agrupación, los datos insertados se sobrescriben visiblemente a los existentes, haciendo aparentemente la función de **`UPDATE`**; por otro lado, al hacer un **`UPDATE`** indicando una clave que no existe, se crea una nueva fila con los datos indicados, haciendo similarmente la función de **`INSERT`**.

Sin embargo, el funcionamiento interno de ambas instrucciones es distinto. En Cassandra, cada fila es una secuencia de **celdas**, correspondientes a cada una de las columnas. Sin embargo, adicionalmente a las celdas visibles, existe una celda oculta que actúa como **marcador de fila** y cuya función es activarse o desactivarse bajo ciertas circunstancias. Para que una fila se tenga en cuenta al realizar las consultas, debe tener al menos una celda activa, ignorando aquellas que tienen todas ellas inactivas (incluido el marcador de fila). Es por esto por lo que podemos ver consultas con filas visiblemente vacías; es decir, cuyas celdas visibles están inactivas pero cuyo marcador de fila está activo. El tratamiento de esta celda oculta es la diferencia entre **`INSERT`** y **`UPDATE`**: la función **`INSERT`** siempre crea una nueva fila con los datos indicados y cuyo marcador de fila se activa; sin embargo, si ya existe una fila con la misma clave que la nueva, se desactivan todas sus celdas incluyendo el marcador de fila. **Esto es lo que sucede en este apartado**. Por otro lado, **`UPDATE`** no modifica los marcadores de fila, por lo que se podría actualizar una fila estableciendo todas sus celdas a **`null`** y todavía seguiría apareciendo en las consultas debido a que **`UPDATE`** no ha desactivado el marcador de fila. Además, si se realiza un **`UPDATE`** con una clave no repetida, inserta los nuevos valores en una nueva fila, ¡pero su marcador de fila estará desactivado! Por lo que si se vuelve a realizar un **`UPDATE`** sobre esa misma fila actualizando todas las columnas a **`null`**, ¡esa fila no aparecerá en las consultas porque su marcador de fila está desactivado! Y esto es justo lo contrario de lo que ocurre si se utiliza **`INSERT`** para insertar una nueva fila en lugar de **`UPDATE`**.

Por lo tanto, lo que ha ocurrido es que la instrucción **`INSERT`** sí ha creado una nueva fila con los datos indicados, pero al mismo tiempo **desactiva todas las celdas (incluido el marcador de fila) de la fila existente ya que detecta que la clave es la misma**. Para solucionarlo, la clave debería incluir la columna que indica de qué edición se trata una carrera (`race_date`) o crear una nueva columna con la edición de la carrera (`race_edition`) e incluirla en la clave. De esta forma, el **`INSERT`** detectará que la clave es diferente (misma carrera pero distinta edición) e insertará la nueva fila sin desactivar todas las celdas de la fila existente.

Ejercicio 2: Neo4j (30 %)

Considera el documento que se encuentra en los materiales del curso “Diseño de una base de datos para analizar la actividad de usuarios en Twitter” que describe la implementación de una base de datos en Neo4j. También se necesitará la maquina virtual “Maquina virtual Neo4j” que contiene una instalación de Neo4j que tiene cargada la base de datos descrita anteriormente.

Se pide proporcionar las siguientes consultas en Cypher y los resultados que se obtienen para las siguientes operaciones (todas las consultas tienen el mismo valor):

1. Obtener el número de tweets geolocalizados que han sido escritos desde Madrid y que son réplicas de tweets geolocalizados.

```
MATCH (l:Location {name: "Madrid"}) <-- (r:Reply) --> (g:GeoLocatedTweet)
WHERE r:GeoLocatedTweet
RETURN count(r) AS Tweets;

+--------+
| Tweets |
+--------+
| 3      |
+--------+
```

2. Obtener el porcentaje de tweets geolocalizados escritos con la aplicación foursquare *(respecto al total de tweets geolocalizados)*.

```
MATCH (g:GeoLocatedTweet)
WITH count(g) AS TotalGeoTweets
MATCH (f:GeoLocatedTweet)
WHERE f.app
CONTAINS "foursquare"
WITH count(f) AS FsGeoTweets, TotalGeoTweets
RETURN (toFloat(FsGeoTweets) / TotalGeoTweets) * 100 AS Porcentaje;

+-------------------+
| Porcentaje        |
+-------------------+
| 6.789884345083283 |
+-------------------+
```

3. Seleccionar el cuarto usuario relevante no verificado con lenguaje de perfil = ‘en’ que ha escrito más tweets. Se debe mostrar solo el nombre de usuario y el número de tweets escritos.

```
MATCH (r:RelevantTwitterUser {verified: 0}) --> (l:Language {languageCode:
"en"})
WITH r.userName AS Name, r.statusesCount AS Tweets
ORDER BY Tweets DESC LIMIT 4
MATCH (Name), (Tweets)
WITH collect(Name) AS n, collect(Tweets) AS t
RETURN n[3] AS Nombre, t[3] AS Tweets;

+--------------------------+
| Nombre          | Tweets |
+--------------------------+
| "NigeriaDotCom" | 32162  |
+--------------------------+
```

4. Considerar el usuario relevante que “se auto sigue” (es un follower de sí mismo) con mayor número de seguidores sumando tanto los seguidores de primer nivel y los de segundo nivel. Debemos mostrar el userName y el número total de seguidores.

```
MATCH (r0:RelevantTwitterUser) -[f0:FOLLOWS]-> (r:RelevantTwitterUser) <-
[f:FOLLOWS]- (t:TwitterUser)
WHERE r0.userId=r.userId
WITH count(f) AS SeguidoresPrimerNivel, r
MATCH (r2:RelevantTwitterUser {userId: r.userId}) <-[f1:FOLLOWS]-
(t2:TwitterUser) <-[f2:FOLLOWS]- (t3:TwitterUser)
WITH count(f2) AS SeguidoresSegundoNivel, SeguidoresPrimerNivel, r
RETURN r.userName AS Nombre, SeguidoresPrimerNivel, SeguidoresSegundoNivel,
SeguidoresPrimerNivel+SeguidoresSegundoNivel AS SeguidoresTotales
ORDER BY SeguidoresTotales DESC LIMIT 1;

+-----------------------------------------------------------------------------------------+
| Nombre           | SeguidoresPrimerNivel | SeguidoresSegundoNivel | SeguidoresTotales |
+-----------------------------------------------------------------------------------------+
| "Ignacio Escolar" | 47                   | 1201                   | 1248              |
+-----------------------------------------------------------------------------------------+
```

5. Contar el número de usuarios relevantes con geolocation activada y que han escrito al menos un geolocated tweet desde “Madrid” o “Barcelona”.

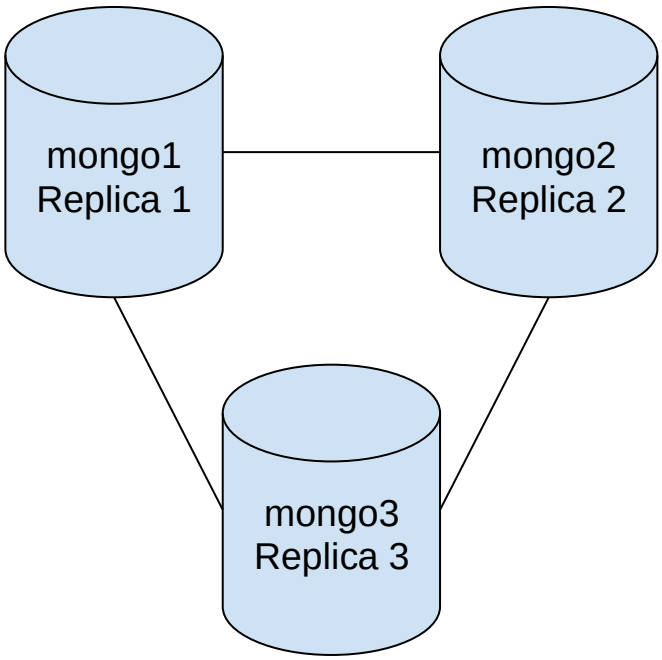
```
MATCH (r:RelevantTwitterUser {geoEnabled: 1}) -[HAS_WRITEN]->
(:GeoLocatedTweet) --> (l:Location)
WHERE l.name="Madrid" OR l.name="Barcelona"
RETURN count(r) AS Usuarios;

+----------+
| Usuarios |
+----------+
| 347      |
+----------+
```

Ejercicio 3: MongoDB consultas (20%)

Para crear la base de datos necesaria para realizar todos los ejercicios de MongoDB, deberéis seguir las instrucciones detalladas en el apartado MongoDB Replicaset disponible en el manual "Uso de máquina virtual".

Cuando terminéis las instrucciones del manual, se estará ejecutando un clúster MongoDB de tres réplicas en la máquina virtual (tres instancias de mongoDB).



En este momento no sabréis cuál es la instancia primaria y cuál es la secundaria hasta que os conectéis a las instancias y/o ejecutéis un comando para ver el estado del replicaset. Un poco más adelante tenéis un enlace con una lista y explicación de los comandos más habituales para gestionar un replicaset en MongoDB.

Tened en cuenta que la elección de un nodo primario depende de muchos factores, la velocidad a la que arranca el nodo, la velocidad de la red, la disponibilidad general de todos los nodos en un momento determinado… **Esto significa que el proceso de elección de un nodo primario no es determinista y cualquier nodo del replicaset puede ser elegido primario cada vez que se arrancan los nodos del clúster.**

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. La base de datos es la base de datos *mensajeria* que hemos estado utilizando hasta ahora.

Para poder realizar algunos de los siguientes ejercicios, necesitaréis la información disponible en esta página del manual online de mongoDB además de los recursos disponibles en el aula. Esta documentación hace referencia a los comandos destinados a trabajar con un replicaset en MongoDB. Tened el enlace "a mano" ya que puede que necesitéis consultarlo.

Ejercicio de ejemplo

En la base de datos de mensajería hay una colección llamada “Student”. Escribir una consulta que retorne todos los documentos de la colección. Por favor, adjuntad la consulta y el resultado en modo texto. Opcionalmente, podéis escribir una breve descripción o justificación de los pasos realizados.

SOLUCIÓN DE EJEMPLO:

Por defecto, mongoDB solamente permite hacer consultas en el nodo primario. Por lo tanto, tendremos que buscar el nodo primario y realizar la consulta en ese nodo o bien podemos ejecutar un comando que permita hacer consultas en el nodo secundario. Optamos por la primera opción que es más fácil. Tal y como se puede comprobar en el *command prompt,* estamos en el nodo primario. El primer comando que utilizaremos será para situarnos en la base de datos “mensajería”. Ejecutamos:

```
Replica_UOC:PRIMARY> use mensajeria
```

Que retorna:

```
switched to db mensajeria
```

Ahora ya podemos realizar la siguiente consulta:

```
Replica_UOC:PRIMARY> db.Student.find()
```

Que retorna:

```
{ "_id" : ObjectId("620f59bb5a7f0e61236102d9"), "Name" : "Marc Cortada" }
```

Este es el formato de respuesta aceptado para este ejercicio. Para realizar los ejercicios normalmente se necesita más de un comando. En tal caso, **se deben adjuntar todos los comandos de manera secuencial junto con su salida o resultado.**

En el ejemplo se ha especificado el comando `use mensajeria` para mostrar un ejemplo con múltiples comandos, pero **para los siguientes ejercicios no es necesario adjuntar el comando `use mensajeria`**. Asumimos que estáis en la base de datos correcta, ya que no sería posible obtener los resultados si estuvierais conectados a otra base de datos.

Si la salida del comando es muy larga y solamente se necesita una parte, podéis acortarla para mostrar solamente la información que habéis usado para resolver el ejercicio.

No es necesario detallar los íconos usados para acceder a la interfaz de comandos, arrancar o parar los servicios.

Sugerencia: la fuente Courier alinea mejor los resultados ya que todas las letras y símbolos tienen el mismo ancho.

Ejercicio 3.1 consulta básica (15%)

La colección “Student” que hay en la base de datos de vuestro entorno es diferente a la que se ha adjuntado como ejemplo. Se pide hacer una consulta que retorne todos los documentos de la colección Student. Para este ejercicio podéis ejecutar la consulta tanto en un nodo primario como secundario. Adjuntar el resultado tal y como se muestra en el ejemplo.

```
db.Student.find()

{ "_id" : ObjectId("628fd93587fb11dbcf9665c2"), "Name" : "Javier Gomez de Diego" }
```

Ejercicio 3.2 consulta (I) (35%)

De la colección *Usuarios_bloqueadores* de la base de datos *mensajeria*, se pide una consulta que muestre todos los usuarios bloqueadores que tengan un usuario bloqueado con el email lizquierdomateo@hotmail.es.

La consulta debe mostrar los siguientes campos: el campo Hash, el campo email del usuario bloqueador (no el email del usuario bloqueado) y debe ocultar el identificador “_id”. Se pide ordenación por email ascendente.

```
db.Usuarios_bloqueadores.find(
      {'Bloqueos.Usuario_bloqueado.Email':'lizquierdomateo@hotmail.es'}
      ,{'Hash':1, 'Email':1, '_id':0}
).sort({'Email':1}).pretty()

{
        "Email" : "cgarciaabril@gmaill.es",
        "Hash" : "nNL2WrqADcvnbajyOcTe6i0xG2SwADGW9oOs8EqtsoDDDGg/lpYgckeF6xbCvoGL/
JqZvP80z9d6wtc4uHo9xor3l20Q9WbegrKzwB43EgrHTgF8MbL2JZYzXtPmEFiWIfeemC/iCtYz2b5AqKt
jQbREB0gH3RgbWoYj48aEFjY="
}
{
        "Email" : "jsanzrobles@hotmail.es",
        "Hash" : "nNL2WrqADcvnbajyOcTe6i0xG2SwADGW9oOs8EqtsoDDDGg/lpYgckeF6xbCvoGL/
JqZvP80z9d6wtc4uHo9xor3l20Q9WbegrKzwB43EgrHTgF8MbL2JZYzXtPmEFiWIfeemC/iCtYz2b5AqKt
jQbREB0gH3RgbWoYj48aEFjY="
}
{
        "Email" : "pcarrillocasa@gmail.es",
        "Hash" : "nNL2WrqADcvnbajyOcTe6i0xG2SwADGW9oOs8EqtsoDDDGg/lpYgckeF6xbCvoGL/
JqZvP80z9d6wtc4uHo9xor3l20Q9WbegrKzwB43EgrHTgF8MbL2JZYzXtPmEFiWIfeemC/iCtYz2b5AqKt
jQbREB0gH3RgbWoYj48aEFjY="
}
{
        "Email" : "vtiernocrespo@hotmail.es",
        "Hash" : "nNL2WrqADcvnbajyOcTe6i0xG2SwADGW9oOs8EqtsoDDDGg/lpYgckeF6xbCvoGL/
JqZvP80z9d6wtc4uHo9xor3l20Q9WbegrKzwB43EgrHTgF8MbL2JZYzXtPmEFiWIfeemC/iCtYz2b5AqKt
jQbREB0gH3RgbWoYj48aEFjY="
}
```

Ejercicio 3.3 consulta (II) (50%)

Hacer un recuento del número de veces que están bloqueados los usuarios de la ciudad de Barcelona, que están en el array *Bloqueos* de la colección *Usuarios_bloqueadores*.

La consulta debe retornar el email del usuario bloqueado y el recuento total de veces que está bloqueado. Se pide ordenación por email ascendente.

Se puede asumir que el campo email es un identificador único válido para identificar los usuarios.

Para facilitar la elaboración de la consulta se adjunta el resultado esperado:

```
{ "_id" : "aretiroperez@hotmail.es", "recuento" : 4 }
{ "_id" : "cgarciaabril@gmaill.es", "recuento" : 4 }
{ "_id" : "efrancolopez@gmail.es", "recuento" : 4 }
{ "_id" : "tjazmintablas@hotmail.es", "recuento" : 3 }
```

Pueden haber varias soluciones válidas, así como varias soluciones que retornen los datos que se esperan, pero no de la manera más eficiente posible. Para puntuar el máximo se valorará también la eficiencia.

Para este ejercicio podéis ejecutar la consulta tanto en un nodo primario como secundario.

```
db.Usuarios_bloqueadores.aggregate([
      {$project: {'U':'$Bloqueos.Usuario_bloqueado'}},
      {$unwind: '$U'},
      {$match: {'U.Ciudad':'Barcelona'}},
      {$group: {'_id': '$U.Email', 'recuento': {$sum:1}}},
      {$project: {'_id':'$_id', 'recuento':'$recuento'},
      {$sort: {'_id':1}}])

{ "_id" : "aretiroperez@hotmail.es", "recuento" : 4 }
{ "_id" : "cgarciaabril@gmaill.es", "recuento" : 4 }
{ "_id" : "efrancolopez@gmail.es", "recuento" : 4 }
{ "_id" : "tjazmintablas@hotmail.es", "recuento" : 3 }
```

Ejercicio 4: MongoDB replicaset (20 %)

Ejercicio 4.1 replicación (20%)

Insertar un documento en la colección “Student” con los siguientes atributos y valores. Uno llamado “Name” y con valor “Jose García”. Otro atributo llamado “Grade” y con valor 9.

Una vez insertado el nuevo documento, escribir una consulta que retorne **todos los documentos de la colección desde cada una de las instancias del *replicaset***. Como hay tres instancias, se tendrán que mostrar todos los comandos necesarios para ejecutar la consulta en cada una de ellas. Las tres consultas deberían mostrar que el documento insertado se ha replicado correctamente en todas las instancias del replicaset.

Recordad que si es necesario ejecutar más comandos para poder ejecutar la consulta, también deben adjuntarse como parte de la solución.

```
# mongo mongo2:27019

Replica_UOC:PRIMARY> use mensajeria

switched to db mensajeria

Replica_UOC:PRIMARY> db.Student.insertOne({'Name':'Jose Garcia', 'Grade':9})

{
        "acknowledged" : true,
        "insertedId" : ObjectId("629252a6d6a053bc926a9970")
}

Replica_UOC:PRIMARY> db.Student.find()

{ "_id" : ObjectId("628fd93587fb11dbcf9665c2"), "Name" : "Javier Gomez de Diego" }
{ "_id" : ObjectId("629252a6d6a053bc926a9970"), "Name" : "Jose Garcia", "Grade" : 9 }

Replica_UOC:PRIMARY> exit

bye

# mongo mongo1:27018

Replica_UOC:SECONDARY> use mensajeria

switched to db mensajeria

Replica_UOC:SECONDARY> rs.secondaryOk()
Replica_UOC:SECONDARY> db.Student.find()

{ "_id" : ObjectId("628fd93587fb11dbcf9665c2"), "Name" : "Javier Gomez de Diego" }
{ "_id" : ObjectId("629252a6d6a053bc926a9970"), "Name" : "Jose Garcia", "Grade" : 9 }

Replica_UOC:SECONDARY> exit

bye

# mongo mongo1:27018

Replica_UOC:SECONDARY> use mensajeria

switched to db mensajeria

Replica_UOC:SECONDARY> rs.secondaryOk()

Replica_UOC:SECONDARY> db.Student.find()

{ "_id" : ObjectId("628fd93587fb11dbcf9665c2"), "Name" : "Javier Gomez de Diego" }
{ "_id" : ObjectId("629252a6d6a053bc926a9970"), "Name" : "Jose Garcia", "Grade" : 9 }
```

Ejercicio 4.2 replicación (30%)

Contesta las siguientes preguntas:

En la colección Student ya había un documento y en el ejercicio anterior se ha insertado uno nuevo. A parte de que los valores son diferentes ¿en qué más se diferencian? ¿Sería posible insertar este documento en una base de datos relacional o tradicional sin hacer nada más?

La diferencia se encuentra en la **estructura**. El nuevo documento tiene tres campos (`_id`, `Name`, `Grade`), mientras que el documento que ya estaba en la colección tiene dos (`_id`, `Name`).

En una base de datos relacional **no sería posible** introducir este documento sin hacer una de las siguientes acciones:

- Eliminar el campo `Grade` del nuevo registro para que su estructura encaje con la de la tabla de la base de datos. De esta forma, se perdería el dato correspondiente a dicho campo del nuevo registro.
- Modificar la estructura de toda la tabla para incluir el campo `Grade`. Sin embargo, como se desconocen los datos del nuevo campo para los registros que ya se encuentran en la base de datos, todos ellos quedarían como `NULL` o `NaN` (excepto el del nuevo registro insertado).

Sin realizar algo de lo anterior, la estructura del nuevo registro no encajaría con la de la tabla de la base de datos relacional, por lo que no se podría insertar tal cual.

Ejercicio 4.3 replicación y consistencia (50%)

Para hacer este ejercicio se deben seguir los siguientes pasos.

Primero. Desde cualquier nodo ejecutad el comando `rs.status()` y adjuntad a continuación solamente la sección `"members" : [...]` de la salida. Resaltad en amarillo el nodo primario en el texto de salida del comando rs.status() como se hace en la salida de ejemplo.

```
Replica_UOC:SECONDARY> rs.status()
(…)
        "members" : [
                {
                        "_id" : 0,
                        "name" : "mongo1:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 268,
                        "optime" : {
                                "ts" : Timestamp(1646940409, 1),
                                "t" : NumberLong(6)
                        },
                        "optimeDurable" : {
                                "ts" : Timestamp(1646940409, 1),
                                "t" : NumberLong(6)
                        },
                        "optimeDate" : ISODate("2022-03-10T19:26:49Z"),
                        "optimeDurableDate" : ISODate("2022-03-10T19:26:49Z"),
                        "lastAppliedWallTime" : ISODate("2022-03-10T19:26:49.229Z"),
                        "lastDurableWallTime" : ISODate("2022-03-10T19:26:49.229Z"),
                        "lastHeartbeat" : ISODate("2022-03-10T19:26:56.265Z"),
                        "lastHeartbeatRecv" : ISODate("2022-03-10T19:26:56.265Z"),
                        "pingMs" : NumberLong(0),
                        "lastHeartbeatMessage" : "",
                        "syncSourceHost" : "mongo2:27019",
                        "syncSourceId" : 1,
                        "infoMessage" : "",
                        "configVersion" : 1,
                        "configTerm" : 6
                },
                {
                        "_id" : 1,
                        "name" : "mongo2:27019",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 268,
                        "optime" : {
                                "ts" : Timestamp(1646940409, 1),
                                "t" : NumberLong(6)
                        },
                        "optimeDurable" : {
                                "ts" : Timestamp(1646940409, 1),
                                "t" : NumberLong(6)
                        },
                        "optimeDate" : ISODate("2022-03-10T19:26:49Z"),
                        "optimeDurableDate" : ISODate("2022-03-10T19:26:49Z"),
                        "lastAppliedWallTime" : ISODate("2022-03-10T19:26:49.229Z"),
                        "lastDurableWallTime" : ISODate("2022-03-10T19:26:49.229Z"),
                        "lastHeartbeat" : ISODate("2022-03-10T19:26:56.265Z"),
```

```
                "lastHeartbeatRecv" : ISODate("2022-03-10T19:26:55.604Z"),
                "pingMs" : NumberLong(0),
                "lastHeartbeatMessage" : "",
                "syncSourceHost" : "",
                "syncSourceId" : -1,
                "infoMessage" : "",
                "electionTime" : Timestamp(1646940159, 1),
                "electionDate" : ISODate("2022-03-10T19:22:39Z"),
                "configVersion" : 1,
                "configTerm" : 6
        },
        {
                "_id" : 2,
                "name" : "mongo3:27020",
                "health" : 1,
                "state" : 2,
                "stateStr" : "SECONDARY",
                "uptime" : 271,
                "optime" : {
                        "ts" : Timestamp(1646940409, 1),
                        "t" : NumberLong(6)
                },
                "optimeDate" : ISODate("2022-03-10T19:26:49Z"),
                "lastAppliedWallTime" : ISODate("2022-03-10T19:26:49.229Z"),
                "lastDurableWallTime" : ISODate("2022-03-10T19:26:49.229Z"),
                "syncSourceHost" : "mongo2:27019",
                "syncSourceId" : 1,
                "infoMessage" : "",
                "configVersion" : 1,
                "configTerm" : 6,
                "self" : true,
                "lastHeartbeatMessage" : ""
        }
]
```

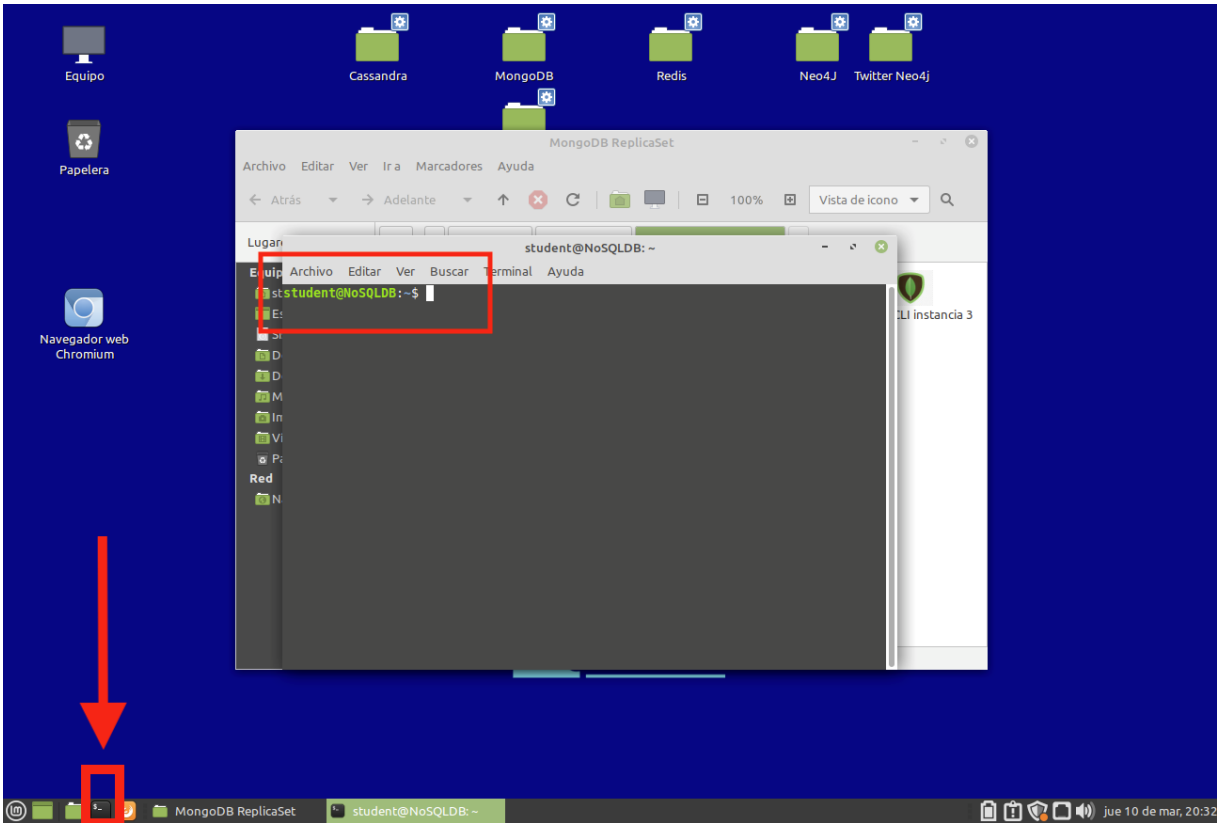
Como podéis observar, en nuestro caso el nodo primario es la máquina o instancia mongo2:27019.

```
"members" : [
        {
                "_id" : 0,
                "name" : "mongo1:27018",
                "health" : 1,
                "state" : 2,
                "stateStr" : "SECONDARY",
                "uptime" : 61072,
                "optime" : {
                        "ts" : Timestamp(1653928064, 1),
                        "t" : NumberLong(14)
                },
                "optimeDurable" : {
                        "ts" : Timestamp(1653928064, 1),
                        "t" : NumberLong(14)
                },
                "optimeDate" : ISODate("2022-05-30T16:27:44Z"),
                "optimeDurableDate" : ISODate("2022-05-30T16:27:44Z"),
                "lastAppliedWallTime" : ISODate("2022-05-30T16:27:44.093Z"),
                "lastDurableWallTime" : ISODate("2022-05-30T16:27:44.093Z"),
                "lastHeartbeat" : ISODate("2022-05-30T16:27:46.317Z"),
                "lastHeartbeatRecv" : ISODate("2022-05-30T16:27:46.659Z"),
```

```
                "pingMs" : NumberLong(0),
                "lastHeartbeatMessage" : "",
                "syncSourceHost" : "mongo2:27019",
                "syncSourceId" : 1,
                "infoMessage" : "",
                "configVersion" : 1,
                "configTerm" : 14
        },
        {
                "_id" : 1,
                "name" : "mongo2:27019",
                "health" : 1,
                "state" : 1,
                "stateStr" : "PRIMARY",
                "uptime" : 334261,
                "optime" : {
                        "ts" : Timestamp(1653928064, 1),
                        "t" : NumberLong(14)
                },
                "optimeDate" : ISODate("2022-05-30T16:27:44Z"),
                "lastAppliedWallTime" : ISODate("2022-05-30T16:27:44.093Z"),
                "lastDurableWallTime" : ISODate("2022-05-30T16:27:44.093Z"),
                "syncSourceHost" : "",
                "syncSourceId" : -1,
                "infoMessage" : "",
                "electionTime" : Timestamp(1653923823, 1),
                "electionDate" : ISODate("2022-05-30T15:17:03Z"),
                "configVersion" : 1,
                "configTerm" : 14,
                "self" : true,
                "lastHeartbeatMessage" : ""
        },
        {
                "_id" : 2,
                "name" : "mongo3:27020",
                "health" : 1,
                "state" : 2,
                "stateStr" : "SECONDARY",
                "uptime" : 61072,
                "optime" : {
                        "ts" : Timestamp(1653928064, 1),
                        "t" : NumberLong(14)
                },
                "optimeDurable" : {
                        "ts" : Timestamp(1653928064, 1),
                        "t" : NumberLong(14)
                },
                "optimeDate" : ISODate("2022-05-30T16:27:44Z"),
                "optimeDurableDate" : ISODate("2022-05-30T16:27:44Z"),
                "lastAppliedWallTime" : ISODate("2022-05-30T16:27:44.093Z"),
                "lastDurableWallTime" : ISODate("2022-05-30T16:27:44.093Z"),
                "lastHeartbeat" : ISODate("2022-05-30T16:27:46.318Z"),
                "lastHeartbeatRecv" : ISODate("2022-05-30T16:27:46.780Z"),
                "pingMs" : NumberLong(0),
                "lastHeartbeatMessage" : "",
                "syncSourceHost" : "mongo2:27019",
                "syncSourceId" : 1,
                "infoMessage" : "",
                "configVersion" : 1,
                "configTerm" : 14}]
```

A continuación, cerrad cualquier ventana abierta del terminal de comandos.

Abrid un nuevo terminal de comandos **que no esté conectado a ningún nodo**. Por ejemplo, podéis usar el icono de la barra de tareas como se muestra en la captura de pantalla. Notad que el terminal de comandos es de la máquina virtual y no está conectado a ningún nodo.



Ejecutad el siguiente comando reemplazando el nombre del nodo primario: `docker network disconnect mongodb_service` **<mongo1|mongo2|mongo3>**

Por ejemplo, si el nodo primario es "name" : "**mongo2:27019**" el comando sería:

```
docker network disconnect mongodb_service mongo2
```

Esto dejará el nodo elegido (en este caso el mongo2) fuera de la red y por tanto fuera del replicaset, lo que simula una caída o una indisponibilidad del nodo en cuestión.

A continuación ejecutad `rs.status()` desde una interfaz de comandos de MongoDB. Es posible que os de algún error y que tengáis que cerrar y volver a abrir la interfaz de comandos. Adjuntad a continuación la sección members[...] como anteriormente y resaltad el nodo que hemos desconectado de la red y el nuevo nodo primario (fijaos que, en este caso, mongo3 ha pasado a ser el nodo primario después de la desconexión de mongo2 de la red).

```
"members" : [
            {
```

```
                        "_id" : 0,
                        "name" : "mongo1:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 5244,
                        "optime" : {
                                "ts" : Timestamp(1646945382, 1),
                                "t" : NumberLong(7)
                        },
                        "optimeDate" : ISODate("2022-03-10T20:49:42Z"),
                        "lastAppliedWallTime" : ISODate("2022-03-10T20:49:42.821Z"),
                        "lastDurableWallTime" : ISODate("2022-03-10T20:49:42.821Z"),
                        "syncSourceHost" : "mongo3:27020",
                        "syncSourceId" : 2,
                        "infoMessage" : "",
                        "configVersion" : 1,
                        "configTerm" : 7,
                        "self" : true,
                        "lastHeartbeatMessage" : ""
                },
                {
                        "_id" : 1,
                        "name" : "mongo2:27019",
                        "health" : 0,
                        "state" : 8,
                        "stateStr" : "(not reachable/healthy)",
                        "uptime" : 0,
                        "optime" : {
                                "ts" : Timestamp(0, 0),
                                "t" : NumberLong(-1)
                        },
                        "optimeDurable" : {
                                "ts" : Timestamp(0, 0),
                                "t" : NumberLong(-1)
                        },
                        "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
                        "optimeDurableDate" : ISODate("1970-01-01T00:00:00Z"),
                        "lastAppliedWallTime" : ISODate("2022-03-10T20:49:22.820Z"),
                        "lastDurableWallTime" : ISODate("2022-03-10T20:49:22.820Z"),
                        "lastHeartbeat" : ISODate("2022-03-10T20:49:48.102Z"),
                        "lastHeartbeatRecv" : ISODate("2022-03-10T20:49:27.320Z"),
                        "pingMs" : NumberLong(0),
                        "lastHeartbeatMessage" : "Error connecting to mongo2:27019
(127.0.1.1:27019) :: caused by :: Connection refused",
                        "syncSourceHost" : "",
                        "syncSourceId" : -1,
                        "infoMessage" : "",
                        "configVersion" : 1,
                        "configTerm" : 7
                },
                {
                        "_id" : 2,
                        "name" : "mongo3:27020",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 5241,
                        "optime" : {
                                "ts" : Timestamp(1646945382, 1),
                                "t" : NumberLong(7)
                        },
```

```
                "optimeDurable" : {
                        "ts" : Timestamp(1646945382, 1),
                        "t" : NumberLong(7)
                },
                "optimeDate" : ISODate("2022-03-10T20:49:42Z"),
                "optimeDurableDate" : ISODate("2022-03-10T20:49:42Z"),
                "lastAppliedWallTime" : ISODate("2022-03-10T20:49:42.821Z"),
                "lastDurableWallTime" : ISODate("2022-03-10T20:49:42.821Z"),
                "lastHeartbeat" : ISODate("2022-03-10T20:49:48.006Z"),
                "lastHeartbeatRecv" : ISODate("2022-03-10T20:49:48.999Z"),
                "pingMs" : NumberLong(0),
                "lastHeartbeatMessage" : "",
                "syncSourceHost" : "",
                "syncSourceId" : -1,
                "infoMessage" : "",
                "electionTime" : Timestamp(1646942151, 1),
                "electionDate" : ISODate("2022-03-10T19:55:51Z"),
                "configVersion" : 1,
                "configTerm" : 7
        }
],
```

```
"members" : [
        {
                "_id" : 0,
                "name" : "mongo1:27018",
                "health" : 1,
                "state" : 2,
                "stateStr" : "SECONDARY",
                "uptime" : 336351,
                "optime" : {
                        "ts" : Timestamp(1653930153, 1),
                        "t" : NumberLong(16)
                },
                "optimeDate" : ISODate("2022-05-30T17:02:33Z"),
                "lastAppliedWallTime" : ISODate("2022-05-30T17:02:33.281Z"),
                "lastDurableWallTime" : ISODate("2022-05-30T17:02:33.281Z"),
                "syncSourceHost" : "mongo3:27020",
                "syncSourceId" : 2,
                "infoMessage" : "",
                "configVersion" : 1,
                "configTerm" : 16,
                "self" : true,
                "lastHeartbeatMessage" : ""
        },
        {
                "_id" : 1,
                "name" : "mongo2:27019",
                "health" : 0,
                "state" : 8,
                "stateStr" : "(not reachable/healthy)",
                "uptime" : 0,
                "optime" : {
                        "ts" : Timestamp(0, 0),
                        "t" : NumberLong(-1)
                },
                "optimeDurable" : {
                        "ts" : Timestamp(0, 0),
                        "t" : NumberLong(-1)
                },
                "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
                "optimeDurableDate" : ISODate("1970-01-01T00:00:00Z"),
                "lastAppliedWallTime" : ISODate("2022-05-30T16:57:23.251Z"),
                "lastDurableWallTime" : ISODate("2022-05-30T16:57:23.251Z"),
                "lastHeartbeat" : ISODate("2022-05-30T17:02:26.325Z"),
                "lastHeartbeatRecv" : ISODate("2022-05-30T16:57:26.323Z"),
                "pingMs" : NumberLong(0),
                "lastHeartbeatMessage" : "Couldn't get a connection within the time
limit",
                "syncSourceHost" : "",
                "syncSourceId" : -1,
                "infoMessage" : "",
                "configVersion" : 1,
                "configTerm" : 16
        },
        {
                "_id" : 2,
                "name" : "mongo3:27020",
                "health" : 1,
                "state" : 1,
                "stateStr" : "PRIMARY",
                "uptime" : 83383,
```

```
                        "optime" : {
                                "ts" : Timestamp(1653930153, 1),
                                "t" : NumberLong(16)
                        },
                        "optimeDurable" : {
                                "ts" : Timestamp(1653930153, 1),
                                "t" : NumberLong(16)
                        },
                        "optimeDate" : ISODate("2022-05-30T17:02:33Z"),
                        "optimeDurableDate" : ISODate("2022-05-30T17:02:33Z"),
                        "lastAppliedWallTime" : ISODate("2022-05-30T17:02:33.281Z"),
                        "lastDurableWallTime" : ISODate("2022-05-30T17:02:33.281Z"),
                        "lastHeartbeat" : ISODate("2022-05-30T17:02:34.501Z"),
                        "lastHeartbeatRecv" : ISODate("2022-05-30T17:02:33.483Z"),
                        "pingMs" : NumberLong(0),
                        "lastHeartbeatMessage" : "",
                        "syncSourceHost" : "",
                        "syncSourceId" : -1,
                        "infoMessage" : "",
                        "electionTime" : Timestamp(1653929713, 1),
                        "electionDate" : ISODate("2022-05-30T16:55:13Z"),
                        "configVersion" : 1,
                        "configTerm" : 16
                }
        ]
```

A continuación, se pide insertar un documento en la colección *Student* con el atributo "Name" y con valor "Pepe Rodríguez". Sugerencia: recordad que si cerráis la interfaz de comandos debéis volveros a situar a la base de datos *mensajeria*, si no estaríais trabajando en la base de datos de *test*.

Una vez insertado el nuevo documento, escribir una consulta que retorne **todos los documentos de la colección desde cada una de las instancias**. Adjuntad los comandos a continuación de este párrafo.
RESPUESTA:

```
Replica_UOC:PRIMARY> db.Student.insertOne({'Name':'Pepe Rodriguez'})

{
        "acknowledged" : true,
        "insertedId" : ObjectId("6294fa7b56e9d2e151aba311")
}

Replica_UOC:PRIMARY> db.Student.find()

{ "_id" : ObjectId("628fd93587fb11dbcf9665c2"), "Name" : "Javier Gomez de Diego" }
{ "_id" : ObjectId("629253dc4b0d86992b4dadfa"), "Name" : "Jose Garcia", "Grade" : 9 }
{ "_id" : ObjectId("6294fa7b56e9d2e151aba311"), "Name" : "Pepe Rodriguez" }

Replica_UOC:PRIMARY> exit

bye

# mongo mongo1:27018

Replica_UOC:SECONDARY> use mensajeria
```

```
switched to db mensajeria

Replica_UOC:SECONDARY> rs.secondaryOk()

Replica_UOC:SECONDARY> db.Student.find()

{ "_id" : ObjectId("628fd93587fb11dbcf9665c2"), "Name" : "Javier Gomez de Diego" }
{ "_id" : ObjectId("629253dc4b0d86992b4dadfa"), "Name" : "Jose Garcia", "Grade" : 9 }
{ "_id" : ObjectId("6294fa7b56e9d2e151aba311"), "Name" : "Pepe Rodriguez" }

Replica_UOC:SECONDARY> exit

bye

# mongo mongo2:27019

Replica_UOC:SECONDARY> use mensajeria

switched to db mensajeria

Replica_UOC:SECONDARY> rs.secondaryOk()

Replica_UOC:SECONDARY> db.Student.find()

{ "_id" : ObjectId("628fd93587fb11dbcf9665c2"), "Name" : "Javier Gomez de Diego" }
{ "_id" : ObjectId("629253dc4b0d86992b4dadfa"), "Name" : "Jose Garcia", "Grade" : 9 }
```

En la interfaz de comandos (o terminal) que no está conectada a ninguna base de datos (la que hemos abierto anteriormente) ejecutad el siguiente comando reemplazando el mismo nodo que habéis desconectado anteriormente (en nuestro caso el comando quedaría de la siguiente manera con el nodo mongo2).

```
docker network connect mongodb_service mongo2
```

Este comando conectará de nuevo la instancia que hemos desconectado anteriormente.

Volved a repetir la consulta de todos los documentos de la colección Student en el nodo que no tenía todos los documentos. Nota: es posible que os de un error y tengáis que cerrar y volver a abrir la ventana de interfaz de comandos. Adjuntad los comandos y el resultado a continuación de este párrafo.
RESPUESTA:

```
# mongo mongo2:27019
Replica_UOC:SECONDARY> use mensajeria

switched to db mensajeria

Replica_UOC:SECONDARY> rs.secondaryOk()

Replica_UOC:SECONDARY> db.Student.find()

{ "_id" : ObjectId("628fd93587fb11dbcf9665c2"), "Name" : "Javier Gomez de Diego" }
{ "_id" : ObjectId("629253dc4b0d86992b4dadfa"), "Name" : "Jose Garcia", "Grade" : 9 }
{ "_id" : ObjectId("6294fa7b56e9d2e151aba311"), "Name" : "Pepe Rodriguez" }
```

Responde la segunda pregunta. Después de la desconexión y conexión de un nodo, ¿el clúster se ha quedado en un estado consistente?
RESPUESTA:

Sí. Tras la conexión, el nodo se ha actualizado y, por tanto, mantiene un estado consistente con el resto de los nodos.