12/11/2020

PRÁCTICA FINAL BASES DE DATOS NO-SQL

ANÁLISIS DE LAS ESTADÍSTICAS DE VUELOS EN ESTADOS UNIDOS (2003 – 2016)

# INTRODUCCIÓN

El objetivo del presente trabajo consiste en analizar las estadísticas de vuelos realizados en los múltiples aeropuertos de Estados Unidos, entre los años 2003 – 2016, (en este último año solo se tienen incluidos los datos del mes de enero). Dicho *dataset*, con un total de 4408 documentos, ha sido extraído del GitHub *awesome-json-datasets[[1]](#footnote-1)* propiedad del usuario **jdorfman**, enlazando a un servidor FTP universitario con ficheros de datos con fines de investigación[[2]](#footnote-2).

## Campos del Dataset

Antes de comenzar con las consultas, analicemos brevemente cada uno de los campos que componen el fichero:

* **Airport**: campo que contiene un objeto JSON, formado por:
  + **Code**: código del aeropuerto, establecido por la Asociación Internacional de Transporte Aéreo o IATA.
  + **Name**: nombre del aeropuerto.
* **Time**: campo que contiene de nuevo un objeto JSON, compuesto por:
  + **Label**: etiqueta con el año y mes en el que se han registrado las estadísticas.
  + **Month**: mes en formato numérico.
  + **Month Name**: mes en formato literal.
  + **Year**: año en formato numérico
* **Statistics**: estadísticas de vuelo registradas en el aeropuerto, mes y año dados. Dicho campo está compuesto por múltiples objetos JSON, los cuales se describen a continuación:
  + **Carriers**:
    - **Names**: *String* que contiene los nombres de compañías aéreas que operaron en el aeropuerto, mes y año dados.
    - **Total**: número total de compañías aéreas, basado en el campo anterior.
  + **# of Delays**:
    - **Carrier**: total de demoras producidos por las aerolíneas.
    - **Late Aircraft**: total de demoras por la tardía llegada de un avión.
    - **National Aviation System**: total de demoras por el sistema nacional de navegación.
    - **Security:** total de demoras por seguridad.
    - **Weather**: total de demoras por cuestiones climáticas.
    - **Total**: total de demoras.
  + **Flights**:
    - **Cancelled**: número de vuelos cancelados.
    - **Delayed**: número de vuelos demorados.
    - **Diverted**: número de vuelos desviados.
    - **On Time**: número de vuelos que llegaron a tiempo.
  + **Minutes Delayed**:
    - **Carrier**: total de minutos demorados por las aerolíneas.
    - **Late Aircraft**: total de minutos demorados por la llegada de un avión.
    - **National Aviation System**: total de minutos demorados por el sistema nacional de navegación.
    - **Security:** total de minutos demorados por seguridad.
    - **Weather**: total de minutos demorados por cuestiones climáticas.
    - **Total**: total de minutos demorados.

# CARGA DEL FICHERO DE DATOS

Una vez descritos los campos, comenzamos con la carga del fichero, mediante la herramienta *mongoimport* de MongoDB, tal y como se muestra a continuación:

Texto

Descripción generada automáticamente

**Ilustración 1**. Carga del fichero de datos JSON

# PREPROCESAMIENTO DE LOS DATOS

**Una vez cargados los datos**, procedemos con una primera consulta, consistente en comprobar el número total de documentos insertados:

db.airports.count()

*>> 4408*

Vemos que el campo *Time* contiene información duplicada, tanto el formato año como el formato mes. Por ello, mediante la función *updateMany* de MongoDB eliminaremos tanto los campos *Time.Label* como *Time.Month Name*, quedándonos con los valores numéricos:

db.airports.updateMany({},{$unset: {"Time.Label": ""}})

db.airports.updateMany({},{$unset: {"Time.Month Name": ""}})

Imagen que contiene Tabla

Descripción generada automáticamente

**Ilustración 2**. Salida updateMany

Como hemos podido comprobar en la descripción de los datos, cada campo es, a su vez, un fichero JSON que contiene información del aeropuerto, e incluso otros objetos JSON como ocurre con el campo *Statistics*. Lo podemos ver con ejemplo de documento que podemos insertar, en el que se puede observar la estructura en objetos JSON contenida en cada documento de la colección (a través de la función *insertOne* de MongoDB):

var aeropuerto = { "Code" : "SLC", "Name" : "Salt Lake City, UT: Salt Lake City International" }

var tiempo = {"Month" : 2, "Year" : 2016 }

var demoras = { "Carrier" : 368, "Late Aircraft" : 549, "National Aviation System" : 253, "Security" : 9, "Weather" : 37 }

var companias = {

"Names" : "American Airlines Inc.,Alaska Airlines Inc.,JetBlue Airways,Delta Air Lines Inc.,Frontier Airlines Inc.,SkyWest Airlines Inc.,United Air Lines Inc.,Southwest Airlines Co.",

"Total" : 8

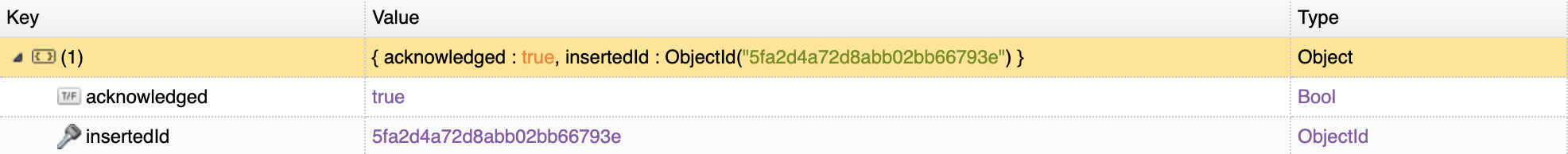
}

var vuelos = { "Cancelled" : 81, "Delayed" : 1170, "Diverted" : 12, "On Time" : 7424, "Total" : 8690 }

var minutos\_demora = { "Carrier" : 32066, "Late Aircraft" : 33682, "National Aviation System" : 8057, "Security" : 57, "Total" : 76978, "Weather" : 3116 }

var estadisticas = { "# of Delays": demoras, "Carriers": companias, "Flights": vuelos, "Minutes Delayed" : minutos\_demora}

db.airports.insertOne({"Airport": aeropuerto, "Time": tiempo, "Statistics": estadisticas})



**Ilustración 3**. Resultado insertOne

Para no afectar a las estadísticas, lo eliminamos por medio de la función *remove*, pasando como condición el campo *\_id*, dado que se trata de un identificador único, evitando con ello borrar otros posibles documentos:

db.airports.remove({"\_id": {$eq: ObjectId("5fa2d4a72d8abb02bb66793e")}})

*>> WriteResult({ "nRemoved" : 1 })*

Por otro lado, dado que el campo *# of Delays* contiene un símbolo no alfa-numérico, lo renombramos a *Delays*, mediante la operación *$rename* del método *updateMany*:

Imagen que contiene Tabla

Descripción generada automáticamente

**Ilustración 4**. Salida updateMany

Imagen que contiene Calendario

Descripción generada automáticamente

**Ilustración 5**. Formato del nuevo campo Delays

Sin embargo, nos seguimos encontrando con un problema: **el campo con las compañías aéreas**, un campo en *String* al que nos gustaría convertirlo, de cara futuras consultas, a un *Array*. Dado que las compañías están separadas por comas, por medio de una función de agregación dividimos el campo *Statistics.Carriers.Names* mediante la operación *$split*. A continuación, por medio de la operación *$addFields* lo añadimos como un campo nuevo con el mismo nombre: *Statistics.Carriers.Names*. Dado que queremos guardar el nuevo campo en disco, mediante la operación *$out* sobreescribimos la colección *airports*:

var companias\_aereas = {$split: ["$Statistics.Carriers.Names", ","]}

var project = {"Statistics.Carriers.Names": companias\_aereas}

var fase1 = {$addFields: project}

var fase2 = {$out: "airports "}

db.airports.aggregate(fase1, fase2)

De este modo, el campo *Statistics.Carriers.Names* contiene cada una de las compañías aéreas en formato Array, tal y como se muestra a continuación:

Texto

Descripción generada automáticamente

**Ilustración 6**. Ejemplo Statistics.Carriers.Names

**Como último apartado de pre-procesamiento**, recordemos que tanto los campos *Carriers*, *Flights* como *Minutes Delayed* contienen un campo *Total*, por lo que antes de trabajar con él vamos a comprobar que se trata, efectivamente, del campo *Total* en cada categoría.

* **Carriers**: para comprobar que el Total de compañías se corresponde con la longitud de cada Array (*Statistics.Carriers.Names*), mediante una operación de agregación comprobaremos en cuántos documentos la longitud del Array corresponde, efectivamente, con lo indicado en el campo *Total*. Para ello, lo dividiremos en tres fases:
  + **Fase 1** (*$project*): contar el número de documentos en el que la longitud del Array corresponde con lo indicado en el campo *Total* (1 o 0):

var condicion = [{$eq: [ {$size: "$Statistics.Carriers.Names"}, "$Statistics.Carriers.Total"]}, 1, 0]

var coincidentes = {"Coincidentes": {$cond: condicion}}

var fase1 = {$project: coincidentes}

* + **Fase 2** (*$group*): tras contar cada documento, sumamos el total de coincidentes:

var group = {\_id: null, SumCoincidentes: {$sum: "$Coincidentes"}}

var fase2 = {$group: group}

* + **Fase 3** (*$project*): dejamos sin mostrar el campo *\_id*:

var fase3 = {$project: {\_id: 0}}

db.airports.aggregate([fase1, fase2, fase3])

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Ilustración 7**. Salida función aggregate

**Como podemos comprobar, el número de filas cuya longitud del Array equivale al campo Total es el mismo al número de documento: 4408**.

* **Flights** y **Minutes Delayed**: para ambos campos se ha creado una función en *JavaScript* con el que podremos comprobar ambos campos. Sobre dicha función pasaremos como parámetros el nombre del campo **raíz** con los valores estadísticos (*$Statistics.Flights* y *$Statistics.Minutes*); los nombres de cada valor (**clave**) situado dentro de cada campo, así como el campo que contiene el total (**campo\_total**):

function comprobar\_total(raiz, claves, campo\_total)

Para ello, la función de agregación contenida se divide en tres fases:

* + **Fase 1** (*$project*): inicialmente, concatenamos (mediante la función *JavaScript forEach*) la raíz junto con los nombres de cada campo. A continuación, mediante la operación *$sum* sumamos cada campo contenido para, posteriormente, compararlo con el campo total, devolviendo un 1 o un 0 si son o no iguales:

var array = [];

claves.forEach(clave => array.push(raiz.concat(clave)));

var suma = {$sum: array};

var condicion = [{$eq: [suma , campo\_total]}, 1, 0];

var coincidentes = {"Coincidentes": {$cond: condicion}};

var fase1 = {$project: coincidentes};

* + **Fase 2** (*$group*): una vez evaluado cada documento, sumamos el total de coincidentes:

var group = {\_id: null, SumCoincidentes: {$sum: "$Coincidentes"}};

var fase2 = {$group: group};

* + **Fase 3** (*$project*): finalmente, dejamos sin mostrar el campo *\_id*:

var fase3 = {$project: {\_id: 0}};

return db.airports.aggregate([fase1, fase2, fase3]);

Una vez creada la función, realizamos la prueba con ambos campos:

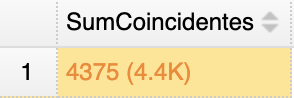
comprobar\_total("$Statistics.Flights.", ["Cancelled", "Delayed", "Diverted", "On Time"], "$Statistics.Flights.Total");

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Ilustración 8**. Salida función aggregate - Statistics.Flights

comprobar\_total("$Statistics.Minutes Delayed.", ["Carrier", "Late Aircraft", "National Aviation System", "Security", "Weather"], "$Statistics.Minutes Delayed.Total");



**Ilustración 9**. Salida función aggregate - Statistics.Minutes Delayed (I)

Podemos comprobar que el campo *Total* de *Statistics.Minutes Delayed* no siempre coincide con la suma de cada uno de sus campos. Como consecuencia, debemos actualizarlo. Para ello, aplicaremos de nuevo una función de agregación, dividida en tres fases:

* + **Fase 1** (*$match*): en primer lugar, buscaremos aquellos documentos en los que el campo *Total* NO coincida con la suma del resto de campos:

var raiz = "$Statistics.Minutes Delayed."

var array\_minutos = [raiz.concat("Carrier"), raiz.concat("Late Aircraft"), raiz.concat("National Aviation System"), raiz.concat("Security"), raiz.concat("Weather")]

var suma = {$sum: array\_minutos}

var condicion = {$ne: [suma, "$Statistics.Minutes Delayed.Total"]}

var fase1 = {$match: {$expr: condicion}}

* + **Fase 2** (*$addFields*): añadimos el nuevo valor, sobreescribiendo el campo *Statistics.Minutes Delayed.Total*:

var total = {"Statistics.Minutes Delayed.Total": suma}

var fase2 = {$addFields: total}

* + **Fase 3** (*$project*): mostramos el campo *\_id* y el campo con el total:

var fase3 = {$project: {\_id:1, total: "$Statistics.Minutes Delayed.Total"}}

Una vez realizada la fase 3, tendremos varios documentos con el campo *\_id* y el campo *total*. Dado que conocemos qué *\_id* debe modificarse, mediante el método *forEach* aplicamos, a cada elemento del *aggregate*, la función *updateMany*, actualizando los documentos correspondientes en disco:

db.airports.aggregate([fase1, fase2, fase3]).forEach(function(id){

db.airports\_modificado.updateMany({"\_id": id.\_id}, {$set: {"Statistics.Minutes Delayed.Total": id.total}})

})

Si volvemos a realizar la llamada a la función que creamos anteriormente, podremos ver que el total de documentos es ahora 4408:

comprobar\_total("$Statistics.Minutes Delayed.", ["Carrier", "Late Aircraft", "National Aviation System", "Security", "Weather"], "$Statistics.Minutes Delayed.Total");

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Ilustración 10**. Salida función aggregate - Statistics.Minutes Delayed (II)

# operaciones crud + aggregate

## 4.1. CONSULTAR EL CODIGO Y NOMBRE DE AEROPUERTO CON EL MAYOR VALOR EN UN CAMPO CONCRETO, EN UN MES Y AÑO EN PARTICULAR

Para ello, creamos una función cuyos parámetros sean el mes, año y campo a consultar, devolviendo el resultado obtenido de una función *find* que filtra los valores del campo a consultar, en el mes y año dados:

function mayor\_valor\_categoria(mes, anno, campo) {

var valor = "$".concat(campo.toString())

var query = {"Time.Year": anno, "Time.Month": { $eq: mes}}

var select = {\_id: 0, "Code": 1, "Airport.Code": 1, "Airport.Name": 1, "Total" : valor}

return db.airports.find(query, select).sort({[campo] : -1]}).limit(1)

}

A modo de ejemplo, **consultamos el aeropuerto con más vuelos realizados en enero del año 2010**, correspondiente al aeropuerto de *Hartsfield-Jackson* de Atlanta:

mayor\_valor\_categoria(1, 2010, "Statistics.Flights.Total")

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

**Ilustración 11**. Salida consulta 1 (I)

Por otro lado, ¿Y si consultamos qué aeropuerto tuvo más vuelos demorados por cuestiones de seguridad en noviembre de 2013?

mayor\_valor\_categoria(11, 2013, "Statistics.Delays.Security")

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

**Ilustración 12**. Salida consulta 1 (II)

Probablemente no resultaría algo llamativo de no ser porque el 5 de noviembre del año 2013 se produjo un tiroteo en el Aeropuerto Internacional de Los Ángeles, lo que probablemente obligaría a cancelar multitud de vuelos[[3]](#footnote-3). Por otro lado, aparte del número de vuelos cancelados ¿Qué habrá ocurrido con el número de vuelos desviados (*Statistics.Flights.Diverted*) en ese mismo mes y año? Para comprobarlo, vamos a modificar la función anterior, limitando el número de salidas a 5 en vez de a 1, para comprobar si el aeropuerto de Los Angeles se encuentra en el top 5 en noviembre del año 2013:

mayor\_valor\_categoria(11, 2013, "Statistics.Flights.Diverted")

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

**Ilustración 13**. Salida consulta 1 (III)

Como podemos comprobar, no solo se encuentra en el top 5, sino que incluso todos los aeropuertos del estado de California (no solo Los Angeles) fueron los que más vuelos desviaron en aquel mes y año, probablemente por cuestiones de Seguridad Nacional.

## 4.2. ANALIZAR EL NUMERO TOTAL DE VUELOS REALIZADOS EN CADA AÑO

Para ello, aplicamos una función de agregación dividida en dos fases, sumando el total de vuelos (*Statistics.Flights.Total*) de cada aeropuerto, agrupando el resultado por año, y finalmente ordenando el resultado por año (*\_id*):

var subquery = {$sum: "$Statistics.Flights.Total"}

var query = {\_id: "$Time.Year", vuelos: subquery}

var fase1 = {$group: query}

var fase2 = {$sort: {"\_id": 1}}

db.airports.aggregate([fase1, fase2])

Imagen que contiene Tabla

Descripción generada automáticamente

**Ilustración 14**. Salida consulta 2 (I)

Incluso, aplicando el método *forEach* a la agregación anterior, podemos consultar la diferencia de vuelos entre dos años consecutivos. Para ello, por cada iteración obtenemos no solo el año del documento sobre el que estamos iterando, sino además el año siguiente, mediante la operación *$match* que recupera los documentos cuyo *\_id* (año) sea mayor. A continuación, volvemos a aplicar las fases 1 y 2 anteriores, agrupando cada documento por la suma total de vuelos ordenados por año, de los cuales se recupera el primer documento. Con ambas parejas, obtenemos la diferencia de vuelos entre ambos años, mostrando finalmente el resultado por pantalla (mediante la función *print*):

db.airports.aggregate([fase1, fase2]).forEach(function(doc) {

var query\_aux = {"$gt": doc.\_id}

var subfase = {$match: {"Time.Year": query\_aux}}

db.airports.aggregate([subfase, fase1,

fase2]).limit(1).forEach(function(doc\_aux) {

var diferencia = doc\_aux.vuelos - doc.vuelos

var texto = ("Diferencia entre " + doc.\_id + " y " + doc\_aux.\_id).toString()

print(texto + ": " + diferencia)

})

})

Texto

Descripción generada automáticamente

**Ilustración 15**. Salida consulta 2 (II)

Esta última consulta nos permite comprobar cómo han evolucionado el número de vuelos con el paso de los años: desde 2003 hasta 2007, el número de vuelos no dejó de aumentar, a un mayor o menor ritmo. Sin embargo, con el estadillo de la crisis económica en el año 2008, el número de vuelos comenzó a reducirse, especialmente durante los primeros años (2008-2012), aunque bien es cierto que hubo periodos (2011-2012 y 2014-2015) en los que el número de vuelos aumentó ligeramente, conforme pasaban los años y poco a poco daba lugar a la recuperación económica, aunque con ciertas caídas (2013-2014). Por otro lado, entre el año 2015 y 2016 se produce una considerable caída en el número de vuelos, aunque esto es debido a la falta de datos, dado que del año 2016 solo se disponen de los datos del mes de enero.

## 4.3. CONSULTAR EL TOP 10 AEROPUERTOS CON EL MAYOR VALOR EN UN CAMPO, JUNTO CON SU PORCENTAJE

Muchas veces, nos interesa realizar una batida de consultas sobre los datos para hacernos a la idea. Por ejemplo, podríamos consultar los 10 aeropuertos donde más vuelos se cancelan, o los 10 aeropuertos donde más vuelos se demoran, o donde más vuelos se demoran a causa del tiempo o cuales son los 10 aeropuertos en los que más vuelos llegan a tiempo. En cualquiera de los casos anteriores, si nos fijamos detenidamente, la consulta es prácticamente la misma: sumar un determinado campo y agruparlo por el nombre de los aeropuertos, ordenando el resultado y filtrando los 10 primeros. Dado que la estructura es la misma, podemos crear una función en *JavaScript* denominada **aeropuerto\_mes\_anno**, en el que pasamos por parámetro el campo a clasificar (Número de vuelos demorados, demorados por tiempo, número de vuelos totales etc.). Para ello, vamos a realizar una función de agregación (dentro de la función anterior) dividida en 7 fases:

function aeropuerto\_mes\_anno(campo)

* **Fase 1** (*$group*): Dado que tenemos que calcular no solo la suma total de un campo por cada aeropuerto sino además su porcentaje con respecto al total, mediante la operación *$group* crearemos un campo *total* que únicamente calcula la suma total del campo pasado como parámetro (SELECT count(\*)). Sin embargo, nos interesa también calcular el total de dicho de campo por aeropuerto, por lo que nos guardaremos para ello la colección original (*$$ROOT*) en un nuevo campo, mediante la operación *$push*

var fase1 = {$group: {\_id: null, total: {$sum: campo}, aeropuerto: { $push: "$$ROOT" }}}

* **Fase 2** (*$unwind*): Ya tenemos el total calculado, junto con la colección original almacenada en un Array ¿Y ahora qué hacemos? Como ya tenemos la suma total del campo, si descomponemos el array con la colección original (*unwind*), cada documento ya tendrá asociado el campo *total*:

var fase2 = {$unwind: "$aeropuerto"}

* **Fase 3** (*$project*): una vez descompuesto el array, nos quedaremos únicamente con aquellos campos que nos interesen, esto es, *\_id* y nombre del aeropuerto, la colección con las estadísticas y el campo *total* previamente calculado:

var fase3 = {$project: {\_id: "$aeropuerto.\_id", "Airport": "$aeropuerto.Airport", "Statistics": "$aeropuerto.Statistics", "total": "$total"}}

* **Fase 4** (*$group*): una vez filtradas las columnas, nos queda agrupar el campo por el nombre del aeropuerto. Por medio de la operación *$group* calculamos el total del campo, aunque esta vez los agrupamos por el nombre del aeropuerto. Además, mediante la operación *$first* recuperamos el campo *total* previamente calculado. Si nos fijamos, con esta fase ya tendríamos la suma total agrupada por cada aeropuerto y el total en general, por lo que nos quedará únicamente ordenar y calcular el porcentaje:

var group = {\_id: "$Airport.Name", vuelos: {$sum: campo}, total: {$first: "$total"}}

var fase4 = {$group: group}

* **Fase 5** (*$sort*): una vez agrupados por cada aeropuerto, ordenamos la colección de mayor a menor:

var fase5 = {$sort: {"vuelos": -1}}

* **Fase 6** (*$addFields*): nos queda calcular el porcentaje, empleando para ello la operación *$divide* entre el total por aeropuerto y el campo *total*, multiplicando el resultado por 100 (*$multiply*), añadiendo un nuevo campo a la colección con el resultado:

var fase6 = {$addFields: field}

var porcentaje = {"$multiply": [{"$divide": ["$vuelos", "$total"]}, 100]}

var field = {"porcentaje (%)": porcentaje}

* **Fase 7** (*$project*): finalmente, dejamos sin mostrar el campo *total*, ya que disponemos del campo con el porcentaje:

var fase7 = {$project: {"total": 0}}

return db.airports.aggregate([fase1, fase2, fase3]).limit(10)

1. https://github.com/jdorfman/awesome-json-datasets [↑](#footnote-ref-1)
2. https://think.cs.vt.edu/corgis/datasets/json/airlines/airlines.json [↑](#footnote-ref-2)
3. https://edition.cnn.com/2013/11/04/justice/lax-shooting/index.html [↑](#footnote-ref-3)