12/11/2020

PRÁCTICA FINAL BASES DE DATOS NO-SQL

ANÁLISIS DE LAS ESTADÍSTICAS DE AEROPUERTOS 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

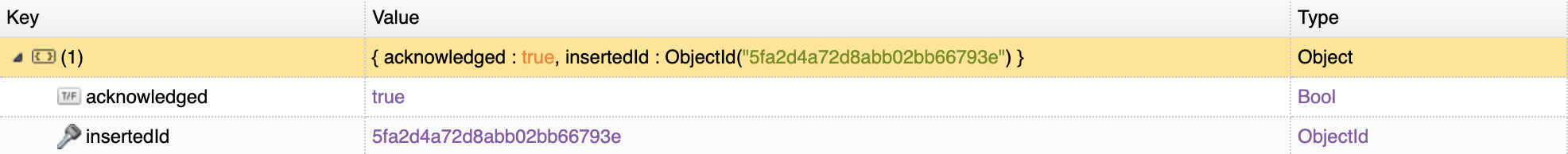
}

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:

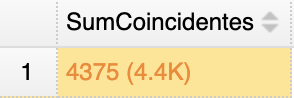
comprobar\_total("$Statistics.Flights.", ["Cancelled", "Delayed", "Diverted", "On Time"], "$Statistics.Flights.Total");

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

Podemos comprobar que el campo *Total* de *Statistics.Minutes Delayed* no siempre coincide con la suma de cada uno de sus campos. Por otro lado, si comparamos la suma de vuelos demorados por cada causa en *Statistics.Delays* con el total de vuelos demorados (*Statistics.Flights.Delayed*) el total tampoco coincide:

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

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Ilustración 10**. Salida función aggregate - Statistics.Flights.Delayed

Como consecuencia, debemos actualizar ambos campos para poder ser utilizados en posteriores agregaciones. 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 de los campos, en al menos uno de los dos casos (*Statistics.Flights.Delayed* o *Statistics.Minutes Delayed*):

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 raiz = "$Statistics.Delays."

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

var suma\_delays = {$sum: array\_delays}

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

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

* + **Fase 2** (*$addFields*): añadimos los nuevos valores, sobreescribiendo los nombres originales de cada campo:

var total = {"Statistics.Minutes Delayed.Total": suma, "Statistics.Flights.Delayed": suma\_delays}

var fase2 = {$addFields: total}

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

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

Dado que conocemos qué *\_id* debe modificarse en la colección original, 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.updateMany({"\_id": id.\_id}, {$set: {"Statistics.Minutes Delayed.Total": id.total\_minutes, "Statistics.Flights.Delayed": id.total\_delays})

})

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

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Ilustración 11**. Salida función aggregate

# 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 12**. 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 13**. 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 14**. 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 15**. 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 16**. 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, limitando el resultado de la agregación a 10 documentos:

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

return db.airports.aggregate([fase1, fase2, fase3, fase4, fase5, fase6, fase7]).limit(10)

Una vez definida la función, ya podemos realizar las consultas que ya hemos mencionado. Por ejemplo, los 10 aeropuertos en los que más vuelos se han cancelado:

aeropuerto\_mes\_anno("$Statistics.Flights.Cancelled")

Tabla

Descripción generada automáticamente

**Ilustración 17**. Top 10 aeropuertos con más vuelos cancelados

Si, adicionalmente, consultamos los 10 aeropuertos con más llegadas a tiempo:

aeropuerto\_mes\_anno("$Statistics.Flights.On Time")

Tabla

Descripción generada automáticamente

**Ilustración 18**. Top 10 aeropuertos con más llegadas a tiempo

De las dos consultas anteriores, caben destacar los tres primeros aeropuertos (Chicago, Atlanta y Dallas), los cuales se mantienen a la cabeza con más vuelos cancelados (con más de un 8 %), pero además con más vuelos a tiempo. No obstante, podemos ver que el porcentaje con respecto al total es mayor para los vuelos cancelados. A modo de ejemplo, mientras que en el aeropuerto de Chicago se cancela un 15 % todos los vuelos cancelados en el país, de todos los vuelos *puntuales*, tan solo el 7,5 % corresponden al aeropuerto de Chicago.

Por otro lado, ¿Y si consultamos qué aeropuertos tienen el mayor número de vuelos demorados? ¿Se obtendrá un top similar a los vuelos cancelados?

aeropuerto\_mes\_anno("$Statistics.Flights.Delayed")

Tabla

Descripción generada automáticamente

**Ilustración 19**. Top 10 aeropuertos con más vuelos demorados

Efectivamente, tanto el aeropuerto de Atlanta como el de Chicago y Dallas encabezan la lista de los aeropuertos con más vuelos demorados, con un porcentaje del 9.9, 9.3 y 6.2 %, respectivamente, con respecto al total. Todas las consultas realizadas hasta ahora nos indican, sumando el total en cada uno, que los aeropuertos de Atlanta, Chicago y Dallas parecen ser los más concurridos entre 2003 y 2016, lo que supone también su problemática como hemos podido comprobar, ya que implica un mayor número de cancelaciones al operar más vuelos con respecto al resto de aeropuertos.

Sin embargo, en muchas ocasiones las demoras o las cancelaciones pueden deberse a varias cuestiones, entre ellas el clima, un factor esencial en el mundo de la aviación. Vamos a estudiar, en relación con la última consulta, qué aeropuertos ha sufrido un mayor número de demoras por causas climatológicas:

aeropuerto\_mes\_anno("$Statistics.Delays.Weather")

Tabla

Descripción generada automáticamente

**Ilustración 20**. Top 10 aeropuertos con más demoras por causas climatológicas

Con respecto al top 3, no hay apenas diferencia en relación con las consultas anteriores. Sin embargo, lo que más interesa no son los tres primeros aeropuertos, sino más bien el resto del top. Si nos fijamos, muchos de los aeropuertos que aparecen corresponden con ciudades del Norte de Estados Unidos: Atlanta, Chicago, Nueva York, Denver, Newark o Boston, entre otros. En estos estados, junto con el resto de los estados del norte/centro de Estados Unidos, destacan las intensas tormentas de nieve y olas de frío que se desatan entre los meses de diciembre y marzo, tormentas que obligan en muchas ocasiones a cancelar o desviar multitud de vuelos.

Incluso si consultamos qué aeropuertos acumulan más minutos de demora por causas climatológicas, el top se mantiene prácticamente idéntico, aunque bien es cierto que hay aeropuertos como en Detroit o en Philadelphia, donde no se han demorado muchos vuelos por causas climatológicas, pero en aquellos que sí, el tiempo en el que han permanecido estacionados habrá sido mayor.

Tabla

Descripción generada automáticamente

**Ilustración 21**. Top 10 aeropuertos con más minutos de demora acumulados por causas climatológicas

## 4.4. CONSULTAR EL MES QUE MAS CANCELACIONES SE HAN REALIZADO DE MEDIA POR CADA AEROPUERTO

En las consultas anteriores hemos comprobado que la mayoría de los aeropuertos en los que más cancelaciones se producen corresponden con ciudades del norte del país, del que habíamos deducido que en muchos casos podría deberse a las grandes tormentas de nieve que se generan anualmente, obligando a cancelar muchos de los vuelos. Ahora bien, pese a que la teoría pueda tener lógica, debemos demostrarlo. Para ello, vamos a consultar en qué mes se realiza un mayor número medio de cancelaciones, mediante el *framework* de agregación que ofrece MongoDB, dividiendo la consulta en 7 fases:

* **Fase 1** (*$group*): en primer lugar, debemos calcular la media por cada aeropuerto, pero cuidado, de cada aeropuerto debemos escoger el mes con mayor media. Para ello, la primera fase consistirá en agrupar la media de vuelos cancelados (*$avg*) tanto por aeropuerto (*Airport.Name*) como por mes (*Time.Month*):

var ids = {id\_aeropuerto: "$Airport.Name", id\_mes: "$Time.Month"}

var media = {$avg: "$Statistics.Flights.Cancelled"}

var query1 = {\_id: ids, media\_cancelaciones: media}

var fase1 = {$group: query1}

* **Fase 2** (*$group*): ya tenemos la media de cancelaciones agrupadas por parejas aeropuerto,mes. Sin embargo, tenemos varios documentos con las medias de cancelaciones por cada aeropuerto y mes, por lo que juntamos estos últimos mediante el operador *$push*, comprimiendo el mes y valor medio (previamente redondeado a dos decimales) de cada aeropuerto dentro un Array, agrupando los resultados por el nombre del aeropuerto (*\_id*):

var redondeo = {$round: ["$media\_cancelaciones", 2]}

var push = {$push: {mes: "$\_id.id\_mes", media\_cancelaciones: redondeo}}

var group = {\_id: "$\_id.id\_aeropuerto", parejas: push}

var fase2 = {$group: group}

* **Fase 3** (*$unwind*): una vez agrupados cada mes y media de cancelaciones en un Array, de cada aeropuerto debemos elegir el mes con mayor media. Para ello, por medio del operador *$unwind* desagregamos la colección. De este modo, podremos posteriormente ordenar la colección de documentos por el campo *media\_cancelaciones*:

var fase3 = {$unwind: "$parejas"}

* **Fase 4** (*$sort*): una vez desagregada la colección, la ordenamos de forma ascendente en función del campo *media\_cancelaciones*:

var fase4 = {$sort: {"parejas.media\_cancelaciones": -1}}

* **Fase 5** (*$group*): tras ordenar la colección, debemos preguntarnos ¿Cómo escogemos el mes con mayor media de cada aeropuerto? Dado que el campo *\_id* empleado en la fase 2 corresponde con el nombre del aeropuerto, además de que la colección está ordenada de forma ascendente, bastará con escoger (agrupando por el nombre de aeropuerto) la media de cancelaciones del primer documento recuperado (*$first*):

var nuevo\_group = {\_id: "$\_id", parejas: {$first: "$parejas"}}

var fase5 = {$group: nuevo\_group}

Para poder mostrar el resultado en un formato de tabla, se han añadido dos fases adicionales (con las cinco anteriores bastaría para obtener el resultado):

* **Fase 6** (*$replaceWith*): para mostrar el resultado en formato tabla, debemos mezclar tanto el campo *\_id* como el contenido del campo *parejas* (*mes*, *media\_cancelaciones*). Para ello, mediante el operador *$mergeObjects* mezclamos ambos campos en un único documento. A continuación, reemplazamos cada documento original (mediante el operador *$replaceWith*) por el nuevo documento:

var merge = [ { aeropuerto: "$\_id"}, "$parejas" ]

var fase6 = { $replaceWith: { $mergeObjects: merge } }

* **Fase 7** (*$sort*): finalmente, ordenamos la colección en función de la media de cancelaciones:

var fase7 = {$sort: {"media\_cancelaciones": -1}}

db.airports.aggregate([fase1, fase2, fase3, fase4, fase5, fase6, fase7])

Tabla

Descripción generada automáticamente

**Ilustración 22**. Listado de meses con mayor media de cancelaciones por aeropuerto

El resultado anterior nos arroja información bastante interesante: **la mayoría de las cancelaciones** (prácticamente en todos los aeropuertos del país) **se produce entre los meses de diciembre y febrero**, es decir, a lo largo del invierno; siendo los aeropuertos del norte como Chicago, Atlanta, Dallas, Nueva York, Boston, Denver o Washington D.C donde más cancelaciones se producen de media. Esto último nos permite confirmar lo anteriormente supuesto: **no sólo se produce un mayor número de cancelaciones (de media) en los meses de invierno, sino que además en los aeropuertos del norte, dadas las intensas tormentas de nieve que se producen y a la elevada demanda de pasajeros durante las vacaciones de Navidad, son donde más cancelaciones se realizan**.

En lugar de agruparlos por aeropuertos, podemos incluso calcular la media por cada mes, empleando solo dos fases:

var fase1 = {$group: {\_id: "$Time.Month", media\_cancelaciones: {$avg: "$Statistics.Flights.Cancelled"}}}

var fase2 = {$sort: {"media\_cancelaciones": -1}}

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

Tabla

Descripción generada automáticamente

**Ilustración 23**. Salida consulta media de cancelaciones por mes

Nuevamente, vemos que tanto el mes de febrero como enero y diciembre están a la cabeza de los meses con mayor media de cancelaciones, coincidencia tanto con los meses de invierno como durante el periodo navideño.

## 4.5. OBTENER EL AEROPUERTO CON MAYOR Y MENOR NÚMERO DE MINUTOS DE DEMORA POR CADA VUELO DEMORADO

Por otro lado, uno de los peores escenarios en un aeropuerto, sin duda, es cuando un vuelo es demorado, lo que obliga a los afectados a tener que soportar interminables esperas en la terminal de varios minutos, incluso horas, hasta verse obligados a cancelar sus billetes y reemplazarlo por un nuevo vuelo, incluso en el mejor de los casos ya que muchas compañías aéreas, especialmente *low-cost*, no garantizan la devolución del importe total.

Como consecuencia, nos gustaría analizar cuál es aeropuerto en Estados Unidos con la mayor y menor proporción *minutos\_demora* / *vuelo\_demorado*. Para ello, emplearemos el *framework* de agregación de MongoDB, a través de un total de 5 fases:

* **Fase 1** (*$group*): inicialmente, debemos calcular el número total de demoras, así como el total de minutos demorados por cada aeropuerto, con el fin de realizar la división en la siguiente fase. Para ello, disponemos del campo *Statistics.Flights.Delayed*, el cual hemos actualizado previamente, con el que obtener el total de vuelos demorados por cada aeropuerto. Por otro lado, para calcular el total de minutos demorados disponemos del campo *Statistics.Minutes Delayed.Total*. Por tanto, la primera fase consistirá en sumar tanto el campo *Statistics.Flights.Delayed* como *Statistics.Minutes Delayed.Total*, agrupados por el nombre del aeropuerto:

var suma\_delays = {$sum: "$Statistics.Flights.Delayed"}

var suma\_minutes\_delayed = {$sum: "$Statistics.Minutes Delayed.Total"}

var group = {\_id: "$Airport.Name", minutes\_delayed: suma\_minutes\_delayed, delays: suma\_delays }

var fase1 = {$group: group}

* **Fase 2** (*$project*): una vez agrupados el total de demoras y minutos demorados por cada aeropuerto, mediante la operación *$divide* calculamos la proporción *minutos\_demorados* / *total\_demoras*, proyectando el resultado como un nuevo campo:

var division = {$divide: ["$minutes\_delayed", "$delays"]}

var fase2 = {$project: {\_id: 1, "Proportion": division}}

* **Fase 3** (*$sort*): una vez calculada la proporción, ordenamos la colección de mayor a menor, en función de dicho campo:

var fase3 = {$sort: {"Proportion": -1}}

* **Fase 4** (*$group*): tras ordenar la colección, nos viene la siguiente pregunta ¿Cómo escogemos el primer y último elemento? Para ello, debemos recordar que cada documento está agrupado por el nombre del aeropuerto, por lo que podríamos dejar la agrupación intacta (marcando el campo *\_id* a *null*), recuperando de la colección anterior (*$$ROOT*) el primer y último grupo, mediante las operaciones *$first* y *$last*, respectivamente. Dado que la colección está ordenada por el campo *Proportion* de forma ascendente, obtendremos los aeropuertos con mayor y menor proporción.

var nuevo\_group = {\_id: null, "Most\_Proportion": {$first: "$$ROOT"}, "Less\_Proportion": {$last: "$$ROOT"}}

var fase4 = {$group: nuevo\_group}

* **Fase 5** (*$project*): por último, dejamos sin mostrar el campo *\_id*.

var fase5 = {$project: {\_id: 0}}

db.airports.aggregate([fase1, fase2, fase3, fase4, fase5])

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Ilustración 24**. Aeropuertos con mayor y menor proporcion minutos\_demora / vuelos\_demorados

Si recordamos de las consultas anteriores, habíamos podido comprobar como los aeropuertos de Atlanta, Chicago y Dallas encabezaban el top 10 con mayor número de demoras e incluso con mayor número de vuelos cancelados. Sin embargo, ninguno de los tres aeropuertos presenta la mayor proporción *minutos/vuelo*, lo que probablemente signifique que dichos aeropuertos tengan un elevado número de vuelos demorados pero el tiempo de demora es bajo. Como consecuencia, el aeropuerto Newark en *New Jersey* encabeza dicho listado, con una proporción de 68 minutos. Esto significaría que a cada vuelo demorado en dicho aeropuerto entre 2003 y 2016 le correspondería algo más de una hora de retraso. Por el contrario, el aeropuerto de San Diego en California presenta la menor proporción, asociando a cada vuelo demorado algo más de 45 minutos de retraso.

## 4.6. ANALIZAR EL NÚMERO MEDIO DE COMPAÑÍAS AÉREAS QUE HA HABIDO A LO LARGO DE LOS AÑOS

Dejando a un lado las estadísticas de vuelos, vamos a echar un vistazo a un campo muy importante y del que apenas hemos hablado: las **compañías aéreas**. Como en cualquier otro sector empresarial, ya sea el mundo del automóvil, telefonía, internet, videojuegos, las compañías aéreas varían con el paso de los años, cesando sus operaciones debido a una posible bancarrota, fusionándose con otras compañías para compartir no solo beneficios sino además infraestructura, incluso con la aparición de nuevas compañías, especialmente *low-cost*. Se trata de un mundo cambiante, con mayor o menor lentitud con respecto a otros sectores, por lo que nos gustaría consultar cómo ha evolucionado el número medio de compañías aéreas a lo largo de los años. Mediante una función de agregación, calculamos el número medio de compañías aéreas por aeropuerto, gracias al campo *Statistics.Carriers.Total*, ordenando el resultado por año y redondeando la media obtenida a un valor entero:

var fase1 = {$group: {\_id: "$Time.Year", companias: {$avg: "$Statistics.Carriers.Total"}}}

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

var fase3 = {$project: {\_id: "$\_id", companias: {$round: ["$companias", 0]}}}

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

Tabla

Descripción generada automáticamente

**Ilustración 25**. Evolución en el número medio de compañías aéreas

Resulta bastante curioso esta última consulta. Al igual que ocurría con el total de vuelos, el número medio de compañías aéreas creció e incluso se mantuvo desde 2003 hasta la entrada de la crisis económica en el año 2008, donde el número medio no ha dejado de disminuir. Sin embargo, ¿Qué ha sido del resto de compañías áereas que han podido quedarse en el camino? ¿En qué años dejaron de operar? ¿Qué compañías han aparecido o desaparecido entre dos años consecutivos? Con el fin de obtener una información mucho más detallada, vamos a profundizar aún más en esta consulta, analizando qué compañías desaparecieron entre dos años consecutivos.

## 4.7. CONSULTAR QUÉ COMPAÑÍAS HAN DESAPARECIDO ENTRE DOS AÑOS CONSECUTIVOS

Para ello, la función de agregación se compondrá de un total de 11 fases:

* **Fase 1** (*$project*): dado que la función de agregación empleará numerosas fases, con el fin de reducir la carga de trabajo, inicialmente proyectaremos los campos que nos interesen, concretamente el año (*Time.Year*) y el Array con los nombres de las compañías aéreas (el campo *\_id* lo mantenemos):

var fase1 = {$project: {anno: "$Time.Year", companias: "$Statistics.Carriers.Names"}}

* **Fase 2** (*$unwind*): de cada año, tenemos en un documento qué compañías aéreas operaron en cada aeropuerto del país. Sin embargo, no todas las compañías operan en todos los aeropuertos, dado que puede haber algunas (como Atlantic Coast Airlines) cuyo de ámbito de operaciones está limitado a determinadas regiones, ya sea la costa este, centro u oeste del país. Por ello, de cada año debemos obtener el conjunto, la **unión** de todas las compañías que hayan operado en al menos un aeropuerto. Inicialmente, desagregaremos la colección por el campo (Array) *compañías*:

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

* **Fase 3** (*$group*): a continuación, creamos una nueva agrupación por año, añadiendo en un conjunto o *set* los nombres de los aeropuertos (por lo que evitamos repeticiones):

var fase3 = {$group: {\_id: "$anno", companias: {$addToSet: "$companias"}}}

* **Fase 4** (*$project*): para facilitar la agregación, proyectamos la colección como un documento formado por los campos *anno* y *compañías*, formado este último por el conjunto de compañías aéreas:

var fase4 = {$project: {\_id: 0, "companias\_anno": {"anno": "$\_id", "companias": "$companias"}}}

* **Fase 5** (*$sort*): para la siguiente fase, ordenamos la colección por el campo año de forma descendente:

var fase5 = {$sort: {"companias\_anno.anno": 1}}

* **Fase 6** (*$group*): ahora bien, ¿Para que hemos hecho el *project* y *sort* anteriores? El objetivo es comparar los elementos del Array con las compañías aéreas entre dos consecutivos, es decir, entre 2003 y 2004, 2004 y 2005, 2005 y 2006 etc. El problema es que solo tenemos la información de cada año una sola vez, por lo que es imposible generar parejas y comparar. Por tanto, la idea es generar un documento adicional que contenga los datos duplicados de cada año comenzando por el 2004, para que de esta forma podamos generar parejas de años consecutivos. En primer lugar, debemos agrupar todos los años en un único Array, al que denominaremos *companias\_anno*:

var fase6 = {$group: {\_id: null, companias\_anno: {$push: "$companias\_anno"}}}

* **Fase 7** (*$project*): una vez agrupados en un mismo documento, debemos crear un duplicado, con la diferencia de que este. nuevo documento empieza por el año 2004 (*companias\_anno\_menos\_1*). Para ello, hacemos uso de la operación *$slice*, la cual permite devolver un subconjunto de un Array pasado como parámetro. En este caso, el sobconjunto será el Array *companias\_anno* menos el primer elemento (es decir, el año 2003):

var longitud = {$size: "$companias\_anno"}

var fase7 = {$project: {\_id: 0, "companias\_anno": 1, "companias\_anno\_menos\_1": {$slice: ["$companias\_anno", 1, longitud]}}}

* **Fase 8** (*$project*): una vez generado el nuevo documento, nos falta combinar cada uno de los elementos de ambos Arrays. Para ello, emplearemos la operación *$zip* que permite combinar dos *Arrays* pasados como parámetro, creando un documento JSON (*input*) por cada combinación:

var fase8 = {$project: {"parejas": {$zip: {"inputs": [ "$companias\_anno", "$companias\_anno\_menos\_1"]}}}}

* **Fase 9** (*$unwind*): una vez generadas las combinaciones, debemos calcular la diferencia del Array con las compañías aéreas por cada pareja de años. Dado que todas las parejas están contenidas en un mismo Array, para facilitar el resto de las fases desagregamos nuevamente mediante la operación *$unwind*:

var fase9 = {$unwind: "$parejas"}

* **Fase 10** (*$project*): dado que ahora cada pareja de años está contenida en un documento diferente, ya podemos trabajar con el Array de compañías aéreas. A modo de ejemplo, entre los años 2003 y 2004 el objetivo será extraer aquellas compañías aéreas que operaron durante el año 2003, pero que no aparecen en el año 2004 (diferencia de conjuntos). Para ello, de cada pareja extraemos el año y el Array con las compañías aéreas (mediante la operación *$arrayElemAt*):

var elem0 = {anno: {$arrayElemAt: ["$parejas.anno", 0]}, diferencia: {$arrayElemAt: ["$parejas.companias", 0]}}

var elem1 = {anno: {$arrayElemAt: ["$parejas.anno", 1]}, diferencia: {$arrayElemAt: ["$parejas.companias", 1]}}

var fase10 = {$project: {elem0, elem1}}

* **Fase 11** (*$project*): una vez obtenidos ambos campos, por medio de la operación *$setDifference* calculamos la diferencia de conjuntos entre ambos Arrays:

var diferencia = {$setDifference: ["$elem0.diferencia", "$elem1.diferencia"]}

var fase11 = {$project: {"annos": ["$elem0.anno", "$elem1.anno"] , "diferencia": diferencia}}

db.airports.aggregate([fase1, fase2, fase3, fase4, fase5, fase6, fase7, fase8, fase9, fase10, fase11])

Imagen que contiene Tabla

Descripción generada automáticamente

**Ilustración 26**. Compañías que no operaron entre dos años consecutivos

Como podemos observar en la captura anterior, entre dos años consecutivos, el número de compañías aéreas ha ido variando, incluso algunas de ellas han dejado de operar. En algunos casos puede haberse tratado de una demanda baja, por lo que la compañía pudo no haber operado en dicho año en favor de la competencia. No obstante, quisiera remarcar ciertos años:

* **2004-2005**:
  + ***Atlantic Coast Airlines***: se trataba de una aerolínea que formaba parte de un *holding* denominado *Atlantic Coast Holdings*. Sin embargo, sus operaciones cesaron a partir del año 2005.
* **2005 – 2006**:
  + ***America West Airlines***: durante muchos años fue la aerolínea principal en Estados Unidos. No obstante, dada su bancarrota fue adquirida por U.S Airways en el año 2007, aunque en el año 2006 sus operaciones ya eran gestionadas por dicha compañía.
  + ***Independence Air***: una aerolínea de bajo coste, cuyas operaciones cesaron en el año 2006, siendo adquirida por Northwest Airlines ese mismo año.
* **2006 – 2007**:
  + ***ATA Airlines d/b/a ATA***: aerolínea *low-cost* que cesó sus operaciones a partir del año 2007 y su desaparición final en 2008, adquirida ese mismo año por *Northwest Airlines*.
* **2008 – 2009**:
  + ***Aloha Airlines***: compañía de vuelos regional centrada en vuelos en/hacia Hawaii, cuyas operaciones cesaron en el año 2008.
* **2009 – 2010**:
  + ***Northwest Airlines***: durante muchos años fue una de las principales aerolíneas de Estados Unidos hasta su bancarrota en el año 2010, año en el cual fue absorbida por la compañía *Delta Airlines Inc.*
* **2010 – 2011**:
  + ***Pinnacle Airlines Inc.***: una aerolínea perteneciente a un *holding* de empresas que, debido a problemas de cambios de sede y compra de compañías que habían quebrado, pudo no haber operado en el año 2011.
  + ***Comair Inc.***: una aerolínea filial de *Delta Airlines*, cuyas operaciones finalizaron en el año 2011 hasta su desaparición final en el año 2012.
* **2011 – 2012**:
  + ***Atlantic Southeast Airlines***: aerolínea que, por decisiones tomadas en su consejo de administración, acordó fusionarse con la compañía *Express Jet Inc.*, cambiando su nombre a *Sure Jet*, dejando de operar como *Atlantic Southeast Airlines* en el año 2011.
  + ***Continental Air Lines***: fue una de las principales aerolíneas en Estados Unidos hasta que, en el año 2011, decidió fusionarse con el gigante *United Airlines*, año en el que todos sus vuelos comenzaron a realizarse bajo la etiqueta *United*.
* **2013 – 2014**:
  + ***Endeavor Air Inc.***: se trata de una compañía de vuelos regional cuyo origen se remonta a 2013, fundada a partir de la compañía *Pinnacle Airlines Inc.* ya mencionada. Pudo no haber operado en 2014 al ser una compañía recién fundada, lo que habría implicado probablemente cambios logísticos y de personal.
  + ***Pinnacle Airlines Inc.***:en favor de *Endeavor Air Inc.*, *Pinnacle Airlines* cesó sus operaciones en el año 2013.
  + ***Mesa Airlines***: se trata de una aerolínea regional que opera en ciertos estados como Arizona, Texas o Kentucky, entre otros. Probablemente, debido a una baja demanda de vuelos en favor de las grandes aerolíneas como *United Airlines* o *American Airways*, pudo no haber operado en el año 2013.
* **2014 – 2015**:
  + ***AirTran Airways Corporation***: aerolínea *low-cost* cuyas operaciones cesaron en el año 2015, aunque fue adquirida por *Southwest Airlines* en el año 2014.
  + ***American Eagle Airlines***: aerolínea subsidiaria de *American Airlines*, que pasó a denominarse *Envoy Air* a partir del año 2015.
* **2015 – 2016**:
  + ***U.S Airways***: una de las aerolíneas estadounidenses de referencia, cuyas operaciones finalizaron en el año 2015, fusionándose con el gigante *American Airlines Inc.*
  + ***Envoy Air***: aerolínea que continúa existiendo hoy en día y que probablemente, debido a que solo se tienen datos del mes de enero del 2016, seguiría operando en dicho año.

Con el transcurso de los años, podemos observar que el número de compañías aéreas se ha visto reducido por varios motivos, desde compañías que entran en bancarrota a partir de la crisis del año 2008, aerolíneas que cambian de nombre hasta incluso aerolíneas que se fusionan para abarcar un mayor número de clientes. Una vez visto qué compañías han ido desapareciendo entre año y año podríamos preguntarnos qué compañías, por el contrario, se han mantenido a lo largo de 13 años (2003 – 2016).

## 4.8. CONSULTAR QUÉ COMPAÑÍAS SE HAN IDO MANTENIENDO CON EL TRANSCURSO DE LOS AÑOS

Para consultar qué aerolíneas se han mantenido a lo largo de 13 años, podemos aplicar una función de agregación muy similar a la del apartado anterior, aunque con menos fases. Si nos fijamos a continuación, las cinco primeras fases o *pipes* son idénticas con respecto al apartado anterior: **proyectamos** el año y el Array con las compañías aéreas, **desagregamos** por el Array, **agrupamos** el conjunto (*set*) de compañías aéreas por año, **proyectamos** un nuevo campo formado por las parejas *años, compañías* y **ordenamos** la colección de forma ascendente:

var fase1 = {$project: {anno: "$Time.Year" , companias: "$Statistics.Carriers.Names"}}

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

var fase3 = {$group: {\_id: "$anno", companias: {$addToSet: "$companias"}}}

var fase4 = {$project: {\_id: 0, "companias\_anno": {"anno": "$\_id", "companias": "$companias"}}}

var fase5 = {$sort: {"companias\_anno.anno": 1}}

Ahora bien, si lo que queremos es recuperar qué compañías aéreas se han mantenido hasta 2016, necesitaremos aplicar la operación intersección *$setIntersection* entre parejas de Arrays. Sin embargo, para poder realizar dicha operación debemos comenzar por el primer año, por lo que crearemos un campo adicional (*primerasCompanias*) que contendrá el primer Array de la colección ($*first*), es decir, el Array con las compañías aéreas del año 2003:

var group = {\_id: 0, "companias": {"$push": "$companias\_anno.companias"}, "primerasCompanias": {"$first": "$companias\_anno.companias"}}

var fase6 = {$group: group}

Una vez tengamos el primer Array separado en un campo aparte, es momento de utilizar la técnica reduce (*$reduce*), aplicando a cada elemento del Array la operación *$setIntersection*, comprimiendo el resultado final en un único campo, denominado *companias*:

var interseccion = {$setIntersection: ["$$value", "$$this"]}

var reduce = {$reduce: {input: "$companias", initialValue: "$primerasCompanias", in: interseccion}}

var fase7 = {$project: {\_id: 0, "companias": reduce}}

db.airports.aggregate([fase1, fase2, fase3, fase4, fase5, fase6])

Texto

Descripción generada automáticamente

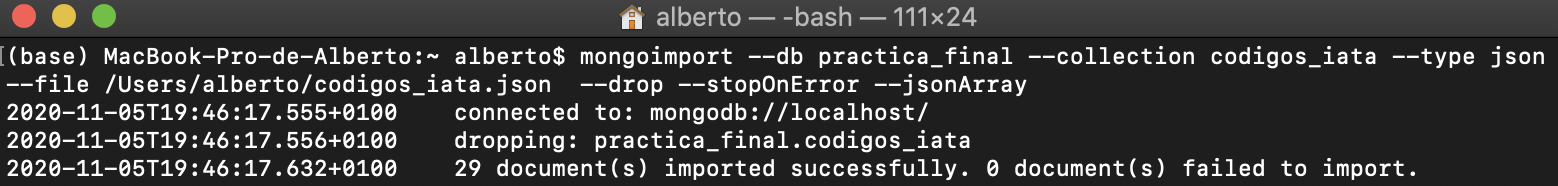
**Ilustración 27**. Compañías que se han mantenido entre 2003 y 2015

No es de extrañar incluso que estas aerolíneas correspondan con las más conocidas no solo a nivel nacional sino incluso internacional (*American Airlines* o *United Air Lines*), compañías que ofrecen decenas de miles de vuelos por todo el mundo y que en muchos casos, para abarcar el mayor número de clientes posibles, anexionan compañías de vuelos tanto a nivel regional como incluso compañías *low-cost* como ya vimos en la consulta anterior.

# CÓDIGOS IATA

Una vez realizadas las consultas anteriores, podríamos consultar el resto de los parámetros que aún no hemos utilizado en el JSON. Sin embargo, quisiera ir un paso más allá, a nivel **geo-espacial**. Como cualquier otra ubicación, un aeropuerto tiene sus coordenadas de latitud y longitud, por lo que ¿Y si juntamos la ubicación de cada aeropuerto con los datos que ya tenemos para poder realizar consulta sobre un mapa? A modo de ejemplo, podríamos consultar los aeropuertos más cercanos a mi ubicación actual (a menos de X metros de distancia). Podríamos ser incluso más ambiciosos y mostrar qué aeropuertos, cercanos a mi ubicación, presenta el menor tiempo medio de demora (*Statistics.Minutes Delayed*). Sin embargo, no disponemos de las coordenadas de cada aeropuerto, por lo que debemos añadirlas. Para ello, utilizaremos otro *dataset* que contiene múltiples campos, y entre otros las coordenadas de latitud y longitud de los aeropuertos anteriores. Dicho *dataset* ha sido recuperado del sitio web oficial *DataHub.io[[4]](#footnote-4)* que contienen varios campos como el nombre del aeropuerto, su código IATA, la altura con respecto al nivel del mar (en pies), además de sus coordenadas de latitud y longitud.

Inicialmente, vamos a cargar el *dataset*, del mismo modo que hicimos con el conjunto de datos inicial (a través del terminal mediante *mongoimport*), creando una colección denominada *códigos\_iata*:



**Ilustración 28**. Carga de datos del fichero codigos\_iata.json

**Nota**: el documento original del sitio web contiene muchos más aeropuertos de los que vamos a utilizar, por lo que de forma previa (a través de un *script* de *Python*) se han eliminado todos aquellos aeropuertos no esenciales, reduciendo considerablemente el tamaño del fichero.

Una vez cargado, echamos un primer vistazo a los datos mediante el método *find*:

db.codigos\_iata.find().limit(1)

Texto

Descripción generada automáticamente

**Ilustración 29**. Ejemplo de un documento de codigos\_iata

Con este documento podemos ver los diferentes campos que componen cada aeropuerto, incluso la altura sobre el nivel del mar de cada aeropuerto, del cual podríamos consultar cuál es el aeropuerto que esté a más de un kilómetro de altura (3280.80 pies):

var sort = {"elevation\_meters": -1}

var condicion = {"elevation\_ft": {$gt: 3280.80}}

var proyeccion = {\_id: 0, "name": 1, "elevation\_ft": 1}

db.codigos\_iata.find(condicion, proyeccion).sort(sort)

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

Descripción generada automáticamente

**Ilustración 30**. Aeropuertos a más de 1 km de altura (3280.80 pies)

Sin embargo, de todos los campos únicamente nos interesa el campo *coordinates*, el cual contiene los valores de latitud y longitud. Sin embargo, nos encontramos con un problema: el campo está en formato cadena. Lo primero que debemos preguntarnos es ¿Cómo es una geo-coordenada o *geo-point* en MongoDB? Un punto en MongoDB debe presentar el siguiente formato:

{

*type*: *Point*,

*coordinates*: [*latitude*, *longitude*]

}

Esto implica que cada coordenada deberá ser un documento JSON con los parámetros mostrados anteriormente, donde *type* indica el formato de la coordenada (en nuestro caso un punto), seguido de las coordenadas en formato Array. Por tanto, mediante una función de agregación crearemos un nuevo campo, al que denominaremos *Position*, formado por el campo *type* como del campo *coordinates*. Para este último, extraeremos de la cadena las coordenadas de latitud y longitud, eliminando posibles espacios en blanco (*$trim*), además de convertir el tipo de dato a *double* (*$toDouble*):

var coordenadas = {$split: ["$coordinates", ","]}

var coord0 = { $arrayElemAt: [ coordenadas, 0 ] }

var coord1 = { $arrayElemAt: [ coordenadas, 1 ] }

var elem0 = {$toDouble: {$trim: {input: coord0}}}

var elem1 = {$toDouble: {$trim: {input: coord1}}}

var posicion = {"Position": {"type": "Point", "coordinates": [elem0, elem1]}}

var fase1 = {$addFields: posicion}

var fase2 = {$out: "codigos\_iata"}

db.codigos\_iata.aggregate([fase1, fase2])

Texto

Descripción generada automáticamente

**Ilustración 31**. Estructura de cada documento con el nuevo campo Position

Como podemos comprobar, ya tenemos las coordenadas en el formato adecuado. Sin embargo, nos falta algo muy importante: **asociar cada coordenada en la colección *codigos\_iata* con las estadísticas de vuelo de la colección anterior**. Para ello, mediante una función de agregación vamos a incluir el campo *Position* de la colección *codigos\_iata* en cada documento con las estadísticas de vuelos, a través de 5 fases:

* **Fase 1** (*$lookup*): probablemente la fase más importante de todas. Lo primero que debemos pensar es ¿Cómo unimos ambas colecciones? Para ello, debemos buscar un campo que tengan en común ambas colecciones por el que se puedan unir como si de una clave primaria-foránea se tratase en SQL. En este caso, ambas colecciones presentan un campo que contiene el código IATA (*International Air Transport Association*):
  + Colección *airports*: *Airport.Code*
  + Colección *codigos\_iata*: *iata\_code*

Para juntar ambas colecciones mediante estos dos campos, emplearemos la operación *$lookup*, que actúa a modo de LEFT OUTER JOIN, la cual nos garantiza que los elementos de la colección en el lado izquierdo aparecerán, independientemente de que aparezcan o no en la colección derecha. Por ello, desde la colección *airports* unimos con la colección *codigos\_iata* empleando tanto el campo local *Airports.Code* como el campo externo o foráneo *iata\_code*, almacenando la información de la segunda colección en un campo denominado *location*:

var parametros = {from: "codigos\_iata", localField: "Airport.Code", foreignField: "iata\_code", as: "location"}

var fase1 = {$lookup: parametros}

* **Fase 2** (*$addFields*): dado que de la colección *iata\_code* sólo nos interesa el campo *Position*, por medio de la operación *$addFields* añadimos un nuevo campo, denominado *Position*, el cual contiene el campo con las coordenadas :

var fase2 = {$addFields: {"Position": "$location.Position"}}

* **Fase 3** (*$set*): sin embargo, el campo *Position* contiene un Array en lugar de un objeto JSON directamente, tal y como se muestra en la siguiente captura:

Imagen que contiene Diagrama

Descripción generada automáticamente

**Ilustración 32**. Campo Position en formato Array

Para evitar esto último, mediante la operación *$arrayElemAt* extraemos el objeto JSON contenido, modificando el campo *Position* mediante la operación *$set*:

var posicion = {$arrayElemAt: ["$Position", 0]}

var fase3 = {$set: {"Position": posicion}}

* **Fase 4** (*$project*): dado que ya tenemos la posición del aeropuerto en un campo aparte, dejamos sin mostrar el campo *location* inicialmente utilizado en la fase 1:

var fase4 = { $project: { location: 0} }

* **Fase 5** (*$out*): finalmente, con la colección ya modificada sobrescribimos la colección original por medio de la operación *$out*:

var fase5 = {$out: "airports"}

db.airports.aggregate([fase1, fase2, fase3, fase4, fase5])

Una vez insertado, mediante *finds* comprobemos que todos los campos existen y, adicionalmente, no estén a null:

db.airports.find({"Position":{ $eq: null}}, {"Position": 1}).count()

*>>> 0*

db.airports.find({"Position":{ $eq: null}}, {"Position": 1}).count()

*>>> 0*

Texto

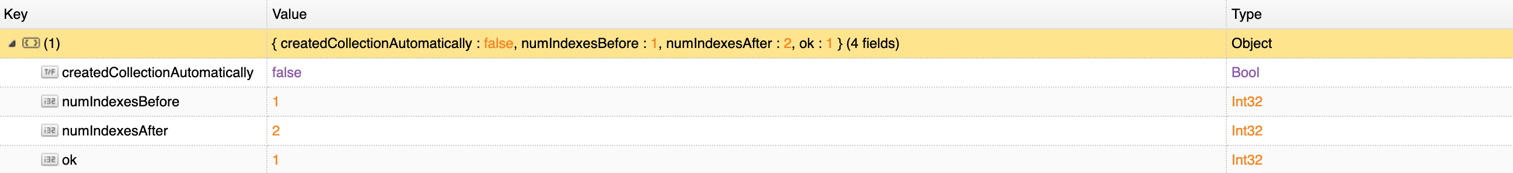
Descripción generada automáticamente

**Ilustración 33**. Colección airports con el nuevo campo Position

Por último, para poder trabajar con las coordenadas es necesario crearse un índice de tipo *2dsphere* con el que poder referenciar al campo *Position* cada vez que se realicen agregaciones con valores de latitud y longitud:

db.airports.getIndexes()

db.airports.createIndex( { Position : "2dsphere" } )



**Ilustración 34**. Salida ejecución createIndex

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)
4. https://datahub.io/core/airport-codes [↑](#footnote-ref-4)