PRÁCTICA FINAL BASES DE DATOS NO-SQL

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

FERNÁNDEZ HERNÁNDEZ, ALBERTO - 12/11/2020

TABLA DE CONTENIDO

1. 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 y 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] propiedad del usuario **jdorfman**, enlazando a un servidor FTP universitario con ficheros de datos con fines de investigación[2].

1.1 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 (IATA).
  - **Name**: nombre del aeropuerto.
- **Time**: campo que contiene 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 la longitud del campo anterior.
  - **# of Delays**:
    - **Carrier**: total de demoras producidas 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 aviación.
    - **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.

[1] https://github.com/jdorfman/awesome-json-datasets
[2] https://think.cs.vt.edu/corgis/datasets/json/airlines/airlines.json

        - **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 aviación.
        - **Security:** total de minutos demorados por seguridad.
        - **Weather**: total de minutos demorados por cuestiones climáticas.
        - **Total**: total de minutos demorados.

2. 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:

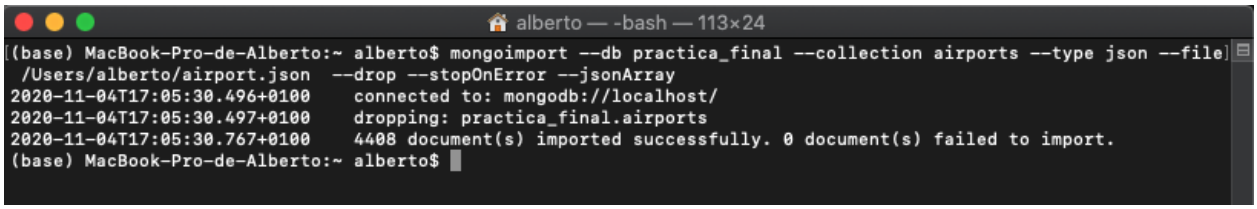

Ilustración 1. Carga del fichero de datos JSON

3. 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

Analizando el contenido de cada documento, vemos que el campo *Time* contiene información duplicada, tanto con el formato año como con 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 de mes y año:

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

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

Key	Value	Type
(1)	{ acknowledged : true, matchedCount : 4408, modifiedCount : 4408 }	Object
acknowledged	true	Bool
matchedCount	4408 (4.4K)	Int32
modifiedCount	4408 (4.4K)	Int32

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, incluso con otros objetos JSON

como ocurre con el campo *Statistics*. Lo podemos ver con un 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})
```

Key	Value	Type
(1)	{ acknowledged : true, insertedId : ObjectId("5fa2d4a72d8abb02bb66793e") }	Object
acknowledged	true	Bool
insertedId	5fa2d4a72d8abb02bb66793e	ObjectId

Ilustración 3. Resultado insertOne

Para no afectar a las estadísticas, lo eliminamos por medio de la función *remove*, pasando como condición el campo mes y año, dado que no hay otro documento con el año 2016 y el mes de febrero (2):

```
db.airports.remove({"Time.Year": 2016, "Time.Month": 2})
```

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

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

```
db.airports.updateMany({}, {$rename: {"Statistics.# of Delays": "Statistics.Delays"}})
```

Key	Value	Type
(1)	{ acknowledged : true, matchedCount : 4408, modifiedCount : 4408 }	Object
acknowledged	true	Bool
matchedCount	4408 (4.4K)	Int32
modifiedCount	4408 (4.4K)	Int32

Ilustración 4. Salida updateMany

```
"Delays" : {
    "Carrier" : 410,
    "Late Aircraft" : 342,
    "National Aviation System" : 312,
    "Security" : 2,
    "Weather" : 27
}
```

Ilustración 5. Formato del nuevo campo *Delays*

Por otro lado, renombramos todos los campos que contengan espacios en blanco, para facilitar las posteriores consultas y agregaciones:

```
// Renombramos los campos que contienen espacios en blanco
db.airports.updateMany({}, {$rename: {
    "Statistics.Flights.On Time":"Statistics.Flights.OnTime",
    "Statistics.Minutes Delayed": "Statistics.MinutesDelayed"
}})

db.airports.updateMany({}, {$rename: {
    "Statistics.MinutesDelayed.Late Aircraft": "Statistics.MinutesDelayed.LateAircraft",
    "Statistics.MinutesDelayed.National Aviation System": "Statistics.MinutesDelayed.NationalAviationSystem",
    "Statistics.Delays.Late Aircraft": "Statistics.Delays.LateAircraft",
    "Statistics.Delays.National Aviation System": "Statistics.Delays.NationalAviationSystem"
}})
```

Ilustración 6. Actualizado de campos con espacios en blanco

Key	Value	Type
(1)	{ acknowledged : true, matchedCount : 4408, modifiedCount : 4408 }	Object
acknowledged	true	Bool
matchedCount	4408 (4.4K)	Int32
modifiedCount	4408 (4.4K)	Int32

Ilustración 7. Salida updateMany

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* sobrescribimos el campo existente: *Statistics.Carriers.Names*. Dado que queremos guardar el nuevo campo en disco, mediante la operación *$out* sobrescribimos 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:

```
"Statistics" : {
    "Carriers" : {
        "Names" : [
            "American Airlines Inc.",
            "Alaska Airlines Inc.",
            "JetBlue Airways",
            "Delta Air Lines Inc.",
            "Frontier Airlines Inc.",
            "Spirit Air Lines",
            "United Air Lines Inc.",
            "Southwest Airlines Co."
        ],
        "Total" : 8
```

Ilustración 8. Ejemplo *Statistics.Carriers.Names*

Como último apartado de preprocesamiento, 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 los 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])
```

	SumCoincidentes
1	4408 (4.4K)

Ilustración 9. Salida función *aggregate*

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

- **Flights** y **Minutes Delayed**: para ambos campos se ha creado una única función en *JavaScript* con la que podremos comprobar ambos campos. Sobre dicha función pasaremos como parámetros el nombre del campo **raíz** con los valores estadísticos (*$Statistics.Flights* y *$Statistics.MinutesDelayed*); los nombres de cada campo (**clave**), 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("$Statistics.Flights.", ["Cancelled", "Delayed", "Diverted", "OnTime"], "$Statistics.Flights.Total");
```

	SumCoincidentes
1	4408 (4.4K)

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

```
comprobar_total("$Statistics.MinutesDelayed.", ["Carrier", "LateAircraft", "NationalAviationSystem", "Security", "Weather"], "$Statistics.MinutesDelayed.Total");
```

	SumCoincidentes
1	4375 (4.4K)

Ilustración 11. Salida función aggregate - Statistics.Minutes Delayed

Podemos comprobar que el campo *Total* de *Statistics.MinutesDelayed* no siempre coincide con la suma de cada uno de sus campos.

Como consecuencia, debemos actualizar dicho campo para poder ser utilizado 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:

```
var raiz = "$Statistics.MinutesDelayed."
var array_minutos = [raiz.concat("Carrier"), raiz.concat("LateAircraft"), raiz.concat("NationalAviationSystem"), raiz.concat("Security"), raiz.concat("Weather")]
var suma = {$sum: array_minutos}
```

```
var condicion = {$ne: [suma, "$Statistics.MinutesDelayed.Total"]}
var fase1 = {$match: {$expr: condicion}}
```

- **Fase 2** (*$addFields*): añadimos los nuevos valores, sobrescribiendo el nombre original:

```
var total = {"Statistics.MinutesDelayed.Total": suma}
var fase2 = {$addFields: total}
```

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

```
var fase3 = {$project: {_id:1, total_minutes: "$Statistics.MinutesDelayed.Total"}}
```

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.MinutesDelayed.Total": id.total_minutes}})
})
```

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

	SumCoincidentes
1	4408 (4.4K)

Ilustración 12. Salida función aggregate

4. 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 del documento a consultar, devolviendo el resultado obtenido a través 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")
```

	Airport		Total
	Code	Name	
1	ATL	Atlanta, GA: Hartsfield-Jackson Atlanta International	33.729 (33.7K)

Ilustración 13. 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")
```

Airport		Total
Code	Name	
LAX	Los Angeles, CA: Los Angeles International	34

Ilustración 14. 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]. 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, aumentando 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")
```

	Airport		Total
	Code	Name	
1	DFW	Dallas/Fort Worth, TX: Dallas/Fort Worth International	76
2	LAX	Los Angeles, CA: Los Angeles International	76
3	SAN	San Diego, CA: San Diego International	56
4	SFO	San Francisco, CA: San Francisco International	55
5	ORD	Chicago, IL: Chicago O'Hare International	36

Ilustración 15. 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 tras el tiroteo.

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 también por año (*_id*):

[3] https://edition.cnn.com/2013/11/04/justice/lax-shooting/index.html

```
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])
```

	_id	vuelos
1	2003	2.373.106 (2.4M)
2	2004	4.344.735 (4.3M)
3	2005	4.373.522 (4.4M)
4	2006	4.437.952 (4.4M)
5	2007	4.538.488 (4.5M)
6	2008	4.307.649 (4.3M)
7	2009	4.038.900 (4.0M)
8	2010	4.050.772 (4.1M)
9	2011	3.923.295 (3.9M)
10	2012	3.955.389 (4.0M)
11	2013	4.108.397 (4.1M)
12	2014	3.824.651 (3.8M)
13	2015	3.870.278 (3.9M)
14	2016	298.634 (0.30M)

Ilustración 16. Salida consulta 2 (I)

Esta última agregación 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 una primera 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, además de calcular el porcentaje de cada uno. Dado que la estructura es la misma, podemos crear una única 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 todas 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 pasado como parámetro, 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 del campo 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 valor por cada 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 hemos mencionado. Por ejemplo, los 10 aeropuertos en los que más vuelos se han cancelado:

```
aeropuerto_mes_anno("$Statistics.Flights.Cancelled")
```

	_id	vuelos	porcentaje (%)
1	Chicago, IL: Chicago O'Hare International	144.426 (0.14M)	15,3422
2	Atlanta, GA: Hartsfield-Jackson Atlanta International	86.176 (86.2K)	9,1544
3	Dallas/Fort Worth, TX: Dallas/Fort Worth International	80.802 (80.8K)	8,5835
4	New York, NY: LaGuardia	57.642 (57.6K)	6,1232
5	Newark, NJ: Newark Liberty International	54.880 (54.9K)	5,8298
6	Boston, MA: Logan International	38.652 (38.7K)	4,106
7	Denver, CO: Denver International	35.926 (35.9K)	3,8164
8	San Francisco, CA: San Francisco International	35.732 (35.7K)	3,7958
9	Los Angeles, CA: Los Angeles International	34.164 (34.2K)	3,6292
10	Houston, TX: George Bush Intercontinental/Houston	30.896 (30.9K)	3,282

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.OnTime")
```

	_id	vuelos	porcentaje (%)
1	Atlanta, GA: Hartsfield-Jackson Atlanta International	3.943.288 (3.9M)	9,6665
2	Chicago, IL: Chicago O'Hare International	3.070.485 (3.1M)	7,5269
3	Dallas/Fort Worth, TX: Dallas/Fort Worth International	2.882.397 (2.9M)	7,0658
4	Denver, CO: Denver International	2.225.040 (2.2M)	5,4544
5	Los Angeles, CA: Los Angeles International	2.212.797 (2.2M)	5,4244
6	Phoenix, AZ: Phoenix Sky Harbor International	1.908.794 (1.9M)	4,6792
7	Houston, TX: George Bush Intercontinental/Houston	1.907.592 (1.9M)	4,6762
8	Las Vegas, NV: McCarran International	1.576.074 (1.6M)	3,8635
9	Detroit, MI: Detroit Metro Wayne County	1.383.667 (1.4M)	3,3919
10	Salt Lake City, UT: Salt Lake City International	1.369.627 (1.4M)	3,3575

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 del país, de todos los vuelos *puntuales*, tan solo el 7,5 % corresponden a dicho aeropuerto.

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")

	_id	vuelos	porcentaje (%)
1	Atlanta, GA: Hartsfield-Jackson Atlanta International	1.052.410 (1.1M)	9,9396
2	Chicago, IL: Chicago O'Hare International	984.968 (0.98M)	9,3027
3	Dallas/Fort Worth, TX: Dallas/Fort Worth International	657.588 (0.66M)	6,2107
4	Los Angeles, CA: Los Angeles International	535.293 (0.54M)	5,0556
5	Denver, CO: Denver International	506.238 (0.51M)	4,7812
6	Newark, NJ: Newark Liberty International	486.592 (0.49M)	4,5957
7	San Francisco, CA: San Francisco International	471.462 (0.47M)	4,4528
8	Houston, TX: George Bush Intercontinental/Houston	432.613 (0.43M)	4,0859
9	Las Vegas, NV: McCarran International	380.875 (0.38M)	3,5972
10	Phoenix, AZ: Phoenix Sky Harbor International	379.748 (0.38M)	3,5866

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. 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")

	_id	vuelos	porcentaje (%)
1	Atlanta, GA: Hartsfield-Jackson Atlanta International	40.113 (40.1K)	11,6344
2	Dallas/Fort Worth, TX: Dallas/Fort Worth International	30.476 (30.5K)	8,8393
3	Chicago, IL: Chicago O'Hare International	24.358 (24.4K)	7,0648
4	New York, NY: LaGuardia	16.350 (16.4K)	4,7422
5	Denver, CO: Denver International	15.556 (15.6K)	4,5119
6	Newark, NJ: Newark Liberty International	14.668 (14.7K)	4,2543
7	Los Angeles, CA: Los Angeles International	14.652 (14.7K)	4,2497
8	Houston, TX: George Bush Intercontinental/Houston	13.062 (13.1K)	3,7885
9	Boston, MA: Logan International	11.955 (12.0K)	3,4674
10	San Francisco, CA: San Francisco International	11.751 (11.8K)	3,4083

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 (a través del campo *Statistics.MinutesDelayed.Weather*), el top 3 se mantiene prácticamente idéntico:

```
aeropuerto_mes_anno("$Statistics.MinutesDelayed.Weather")
```

	_id	vuelos	porcentaje (%)
1	Atlanta, GA: Hartsfield-Jackson Atlanta International	3.209.941 (3.2M)	11,6012
2	Dallas/Fort Worth, TX: Dallas/Fort Worth International	2.549.836 (2.5M)	9,2155
3	Chicago, IL: Chicago O'Hare International	2.192.250 (2.2M)	7,9231
4	Denver, CO: Denver International	1.240.212 (1.2M)	4,4823
5	Houston, TX: George Bush Intercontinental/Houston	1.209.686 (1.2M)	4,372
6	New York, NY: LaGuardia	1.205.884 (1.2M)	4,3583
7	Newark, NJ: Newark Liberty International	1.180.657 (1.2M)	4,2671
8	Los Angeles, CA: Los Angeles International	1.029.615 (1.0M)	3,7212
9	Detroit, MI: Detroit Metro Wayne County	1.025.080 (1.0M)	3,7048
10	Philadelphia, PA: Philadelphia International	949.766 (0.95M)	3,4326

***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 producen 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, por cada aeropuerto, en qué mes se realiza un mayor promedio 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** (*$group*): ya tenemos la media de cancelaciones agrupadas por parejas aeropuerto, mes. Sin embargo, tenemos varios documentos con las medias de cancelaciones en un mismo aeropuerto, por lo que los juntamos mediante el operador *$push*, comprimiendo cada mes y valor medio (previamente

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

```
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 de nuevo 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])
```

	aeropuerto	mes	media_cancelaciones
1	Chicago, IL: Chicago O'Hare International	2	1579,92 (1.6K)
2	Atlanta, GA: Hartsfield-Jackson Atlanta International	1	1120,54 (1.1K)
3	Dallas/Fort Worth, TX: Dallas/Fort Worth International	2	883,5
4	New York, NY: LaGuardia	2	636,83
5	Newark, NJ: Newark Liberty International	2	636,83
6	Boston, MA: Logan International	2	514,83
7	Denver, CO: Denver International	12	468
8	Washington, DC: Ronald Reagan Washington National	2	397,5
9	New York, NY: John F. Kennedy International	2	384,33
10	San Francisco, CA: San Francisco International	12	369,23
11	Detroit, MI: Detroit Metro Wayne County	2	345,83
12	Washington, DC: Washington Dulles International	2	332,08
13	Houston, TX: George Bush Intercontinental/Houston	9	330
14	Los Angeles, CA: Los Angeles International	1	323,54
15	Philadelphia, PA: Philadelphia International	2	316,17
16	Baltimore, MD: Baltimore/Washington International Thurgood Marshall	2	296,83
17	Charlotte, NC: Charlotte Douglas International	1	296,69
18	Chicago, IL: Chicago Midway International	2	267,67
19	Minneapolis, MN: Minneapolis-St Paul International	2	246,08
20	Phoenix, AZ: Phoenix Sky Harbor International	1	239,69
21	Salt Lake City, UT: Salt Lake City International	12	218,54
22	Orlando, FL: Orlando International	2	160,17
23	Las Vegas, NV: McCarran International	1	151,08
24	Seattle, WA: Seattle/Tacoma International	1	140,92
25	Miami, FL: Miami International	8	129,46
26	San Diego, CA: San Diego International	12	113,31
27	Fort Lauderdale, FL: Fort Lauderdale-Hollywood International	2	108,42
28	Portland, OR: Portland International	1	96,23
29	Tampa, FL: Tampa International	2	95,42

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: **se produce un mayor número de cancelaciones (de media) en los meses de invierno, especialmente 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**.

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])
```

	_id	media_cancelaciones
1	2	361,0057
2	1	340,7294
3	12	286,5411
4	3	223,4224
5	7	210,0637
6	6	208,9947
7	8	196,4244
8	4	158,5402
9	9	158,0875
10	5	154,908
11	10	149,3236
12	11	118,0133

Ilustración 23. Salida consulta media de cancelaciones por mes

Nuevamente, vemos que tanto el mes de diciembre como enero y febrero están a la cabeza de los meses con mayor media de cancelaciones, coincidiendo 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 total del importe.

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.MinutesDelayed.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])
```


```
{
    "Most_Proportion" : {
        "_id" : "Newark, NJ: Newark Liberty International",
        "Proportion" : 68.44891202485861
    },
    "Less_Proportion" : {
        "_id" : "San Diego, CA: San Diego International",
        "Proportion" : 46.92722403003755
    }
}
```


Ilustración 24. Aeropuertos con mayor y menor proporción 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 demoras, pero con un tiempo de demora 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 año, gracias al campo *Statistics.Carriers.Total*:

```
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])
```

	_id	companias
1	2003	12
2	2004	13
3	2005	14
4	2006	14
5	2007	14
6	2008	14
7	2009	13
8	2010	13
9	2011	12
10	2012	11
11	2013	11
12	2014	10
13	2015	10
14	2016	9

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 las compañías aéreas entre dos años 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 documentos anteriores 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 (comenzando por la posición 1):

```
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 único Array (*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])
```

	annos	diferencia
1	[2003, 2004]	[ ]
2	[2004, 2005]	["Atlantic Coast Airlines"]
3	[2005, 2006]	["Independence Air", "America West Airlines Inc."]
4	[2006, 2007]	["ATA Airlines d/b/a ATA"]
5	[2007, 2008]	[ ]
6	[2008, 2009]	["Aloha Airlines Inc."]
7	[2009, 2010]	["Northwest Airlines Inc."]
8	[2010, 2011]	["Pinnacle Airlines Inc.", "Comair Inc."]
9	[2011, 2012]	["Atlantic Southeast Airlines", "Continental Air Lines Inc."]
10	[2012, 2013]	[ ]
11	[2013, 2014]	["Endeavor Air Inc.", "Pinnacle Airlines Inc.", "Mesa Airlines Inc."]
12	[2014, 2015]	["AirTran Airways Corporation", "American Eagle Airlines Inc."]
13	[2015, 2016]	["US Airways Inc.", "Envoy Air"]

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 los intervalos de años que aparecen en la agregación:

- **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])
```

```
{
    "companias" : [
        "Alaska Airlines Inc.",
        "American Airlines Inc.",
        "Delta Air Lines Inc.",
        "ExpressJet Airlines Inc.",
        "Hawaiian Airlines Inc.",
        "JetBlue Airways",
        "SkyWest Airlines Inc.",
        "Southwest Airlines Co.",
        "United Air Lines Inc."
    ]
}
```

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 cientos 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.

5. CÓDIGOS IATA

Una vez realizadas las consultas anteriores, quisiera ir un paso más allá, a nivel **geoespacial**. 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 una consulta sobre un mapa? A modo de ejemplo, podríamos consultar los aeropuertos más cercanos a mi ubicación actual (a menos de X kilómetros de distancia). Podríamos ser incluso más ambiciosos y mostrar la distancia a aquellos aeropuertos en todo el país con menor media de minutos de demora (*Minutes Delayed*). Sin embargo, no disponemos de las coordenadas de cada aeropuerto, por lo que debemos añadirlas. Para ello, utilizaremos otro *dataset* que contiene, entre otros campos, las coordenadas de latitud y longitud de los aeropuertos anteriores. Dicho *dataset* ha sido recuperado del sitio web oficial *DataHub.io*[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 *codigos_iata*:

```
alberto — -bash — 111×24
(base) MacBook-Pro-de-Alberto:~ alberto$ mongoimport --db practica_final --collection codigos_iata --type json
--file /Users/alberto/codigos_iata.json  --drop --stopOnError --jsonArray
2020-11-05T19:46:17.555+0100    connected to: mongodb://localhost/
2020-11-05T19:46:17.556+0100    dropping: practica_final.codigos_iata
2020-11-05T19:46:17.632+0100    29 document(s) imported successfully. 0 document(s) failed to import.
```

Ilustración 28. Carga de datos del fichero codigos_iata.json

[4] https://datahub.io/core/airport-codes

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)
```


```
{
    "_id" : ObjectId("5fa4487932c275eae2d13eae"),
    "continent" : "NA",
    "coordinates" : "-71.00520325, 42.36429977",
    "elevation_ft" : 20,
    "gps_code" : "KBOS",
    "iata_code" : "BOS",
    "ident" : "KBOS",
    "iso_country" : "US",
    "iso_region" : "US-MA",
    "local_code" : "BOS",
    "municipality" : "Boston",
    "name" : "General Edward Lawrence Logan International Airport",
    "type" : "large_airport"
}
```


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)
```

	elevation_ft	name
1	5431 (5.4K)	Denver International Airport
2	4227 (4.2K)	Salt Lake City International Airport

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])
```


```
{
    "_id" : ObjectId("5fa4487932c275eae2d13eae"),
    "continent" : "NA",
    "coordinates" : "-71.00520325, 42.36429977",
    "elevation_ft" : 20,
    "gps_code" : "KBOS",
    "iata_code" : "BOS",
    "ident" : "KBOS",
    "iso_country" : "US",
    "iso_region" : "US-MA",
    "local_code" : "BOS",
    "municipality" : "Boston",
    "name" : "General Edward Lawrence Logan International Airport",
    "type" : "large_airport",
    "Position" : {
        "type" : "Point",
        "coordinates" : [
            -71.00520325,
            42.36429977
        ]
    }
}
```


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 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:

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 insertados, mediante *finds* comprobamos que todos los campos *Position* 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


```
{
    "_id" : ObjectId("5fa41afbc8fdc4eab983d252"),
    "Airport" : {
        "Code" : "TPA",
        "Name" : "Tampa, FL: Tampa International"
    },
    "Time" : {
        "Month" : 1,
        "Year" : 2016
    },
    "Statistics" : {
        "Carriers" : {
            "Names" : [
                "American Airlines Inc.",
                "Alaska Airlines Inc.",
                "JetBlue Airways",
                "Delta Air Lines Inc.",
                "Frontier Airlines Inc.",
                "Spirit Air Lines",
                "United Air Lines Inc.",
                "Southwest Airlines Co."
            ],
            "Total" : 8
        },
        "Flights" : {
            "Cancelled" : 146,
            "Delayed" : 1095,
            "Diverted" : 7,
            "On Time" : 4748,
            "Total" : 5996
        },
        "Minutes Delayed" : {
            "Carrier" : 22557,
            "Late Aircraft" : 21164,
            "National Aviation System" : 11423,
            "Security" : 37,
            "Total" : 57774,
            "Weather" : 2593
        },
        "Delays" : {
            "Carrier" : 410,
            "Late Aircraft" : 342,
            "National Aviation System" : 312,
            "Security" : 2,
            "Weather" : 27
        }
    },
    "Position" : {
        "type" : "Point",
        "coordinates" : [
            -82.533203125,
            27.975500106811523
        ]
    }
},
```


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" } )
```

Key	Value	Type
(1)	{ createdCollectionAutomatically : false, numIndexesBefore : 1, numIndexesAfter : 2, ok : 1 } (4 fields)	Object
createdCollectionAutomatically	false	Bool
numIndexesBefore	1	Int32
numIndexesAfter	2	Int32
ok	1	Int32

Ilustración 34. Salida ejecución *createIndex*

Una vez creado el índice, ya podemos realizar agregaciones sobre coordenadas geográficas. En este proyecto, realizaremos dos consultas de prueba.

5.1. CONSULTAR LA DISTANCIA A LOS 3 AEROPUERTOS CON MENOR MEDIA DE MINUTOS DE DEMORA EN CADA CATEGORÍA

Si recordamos del JSON inicial, el campo *Minutes Delayed* estaba compuesto, a su vez, por varios campos:

- *Carrier*
- *Late Aircraft*
- *National Aviation System*
- *Security*
- *Weather*

Por tanto, el objetivo es obtener los 3 aeropuertos (y su distancia a un punto de referencia) con menor media de minutos de demora en cada categoría. Para ello, elaboraremos una función en *JavaScript* que nos permita realizar una función de agregación según el campo a consultar, a la que denominaremos *calcular_distancia*:

```
function calcular_distancia(campo)
```

Dicha función contiene una agregación dividida en tres fases:

- **Fase 1** (*$geoNear*): MongoDB, por su parte, dispone del operador *$geoNear* con la que es posible calcular las distancias entre coordenadas geográficas, e incluso (como se verá en la siguiente consulta) obtener aquellos puntos a menos de X kilómetros de distancia. Para esta consulta, le pasaremos los siguientes parámetros:
  - ***near***: punto de referencia con el que calcular la distancia. Para este ejemplo, tomaremos como referencia una ciudad situada en el centro de Estados Unidos, como por ejemplo Wichita[5] , a través de sus coordenadas de latitud y longitud.
  - ***distanceField***: campo sobre el que se almacenará la distancia calculada (lo denominaremos como *dist.calculated*)
  - ***includeLocs***: almacena en un campo la ubicación empleada para calcular una distancia (la localización de cada aeropuerto), al que denominaremos *dist.location*
  - ***distanceMultiplier***: factor por el que multiplicar la distancia obtenida. Dado que la distancia obtenida está en metros, debemos pasarlo a kilómetros (x 0.001)
  - ***spherical***: determina cómo MongoDB calcula la distancia. Para este ejemplo, utilizaremos geometría esférica (poniendo dicho campo a *true*), es decir, calculando la distancia considerando el escenario como una esfera y no como un plano (no emplea la distancia Euclidiana)

[5] https://es.wikipedia.org/wiki/Wichita_(Kansas)

```
var parametros = {
        near: {type: "Point", coordinates: [ -97.347059 , 37.631635 ] },
        distanceField: "dist.calculated",
        includeLocs: "dist.location",
        distanceMultiplier: 0.001,
        spherical: true
      }
var fase1 = {$geoNear: parametros}
```

- **Fase 2** (*$group*): una vez calculada la distancia entre todos los aeropuertos, mediante el operador *$avg* calculamos la media del campo pasado como parámetro (creando el campo *Minutes Delayed*), agrupando el resultado por cada aeropuerto y su distancia obtenida:

```
var media_minutes_delayed = {$avg: campo}
var group = {_id: {"Location": "$dist.location", "Airport": "$Airport.Name", "Distance": "$dist.calculated"}, "Minutes_Delayed": media_minutes_delayed}
var fase2 = {$group: group}
```

- **Fase 3** (*$sort*): Finalmente, ordenamos la colección resultante por el campo *Minutes_Delayed*, proyectando los tres primeros documentos:

```
var fase3 = {$sort: {"Minutes_Delayed": 1}}
return db.airports.aggregate([fase1, fase2, fase3]).limit(3)
```

Una vez creada la función, es momento de probar con cada campo:

Minutes Delayed.Carrier

```
calcular_distancia("$Statistics.MinutesDelayed.Carrier")
```

_id			Minutes_Delayed
Location	Airport	Distance	
{ type : "Point", coordinates : [-122.5979996, 45.58869934] }	Portland, OR: Portland International	2268,449 (2.3K)	13.215,3026 (13.2K)
{ type : "Point", coordinates : [-87.752403, 41.785999] }	Chicago, IL: Chicago Midway International	942,0446	14.810,25 (14.8K)
{ type : "Point", coordinates : [-82.533203125, 27.975500106811523] }	Tampa, FL: Tampa International	1750,2239 (1.8K)	18.426,7763 (18.4K)

Ilustración 35. Top 3 aeropuertos con menor valor medio de minutos de demora por aerolíneas

Como se puede observar en la captura anterior, MongoDB nos ha calculado la distancia (en kilómetros) a cada aeropuerto, donde podemos comprobar que los aeropuertos de Portland, Chicago y Tampa son aquellos con menor media de minutos de demora por compañías aéreas (*Carrier*). Sin embargo, dado que hemos calculado la distancia y tenemos coordenadas geográficas, lo más adecuado sería mostrarlo en un mapa. Por tanto, tanto esta consulta como las siguientes también se realizarán mediante un *script* de Python el cual se adjuntará en la documentación (denominado *pinta_coordenadas.py*), el cual realizará la misma agregación que se ha mostrado anteriormente, mediante la librería *pymongo*. Una vez obtenido el resultado (en formato diccionario) mediante una librería específica para mapas (*folium*) se mostrará cada punto con la información

obtenida en la agregación a través de un mapa OpenStreetMaps en HTML, con el fin de obtener una respuesta mucho más visual, así como las distancias entre puntos.

Por tanto, si realizamos la misma agregación con el campo anterior, desde Python el mapa nos quedará con el siguiente aspecto:

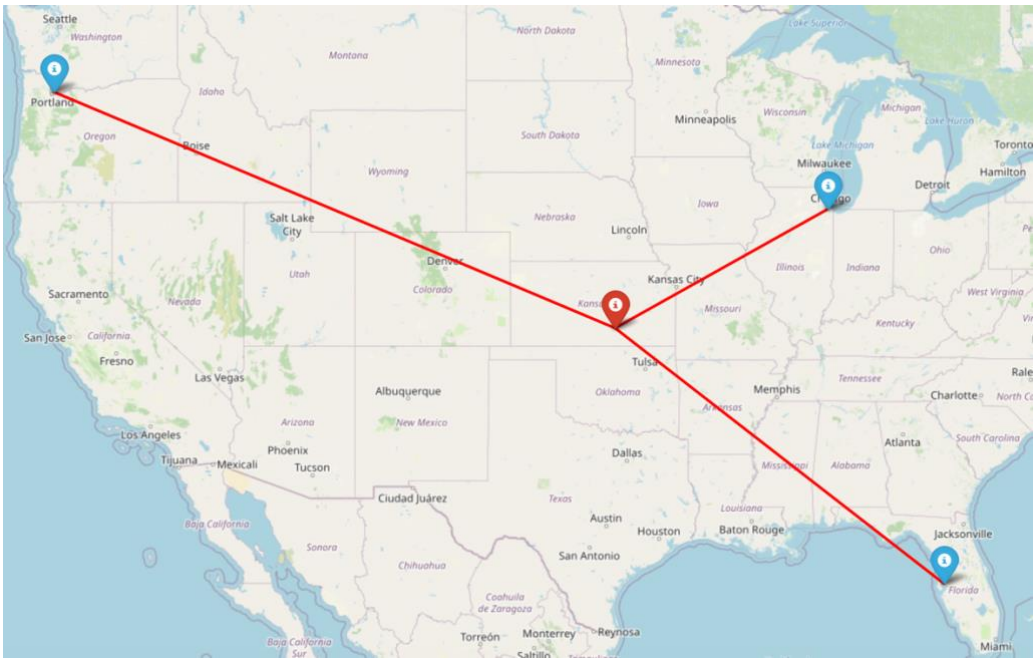

Ilustración 36. Mapa Minutes Delayed – Carrier

Con la información obtenida desde MongoDB, podemos incluso obtener la información de cada aeropuerto:

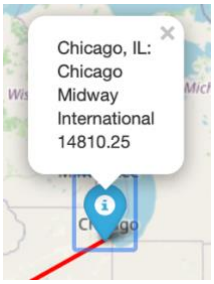

Ilustración 37. Aeropuerto + Media Minutes Delayed.Carrier

E incluso la distancia en kilómetros se puede consultar *pinchando* sobre las líneas rojas, obteniendo la distancia devuelta por el *framework* de agregación:

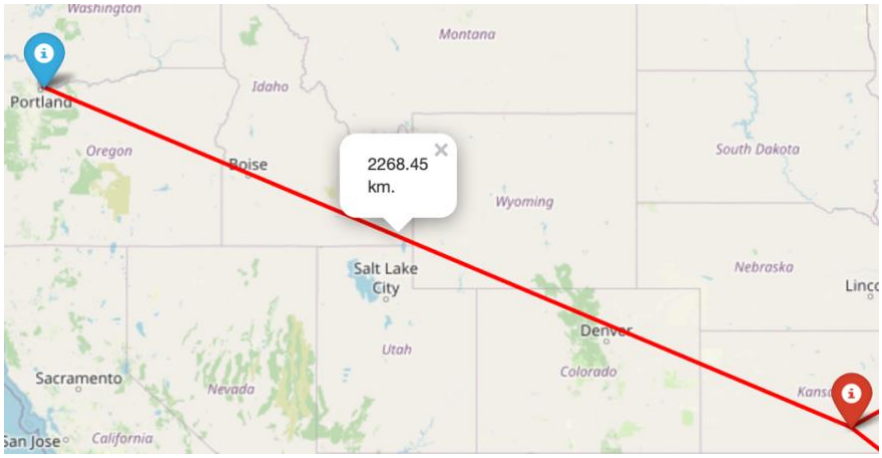

Ilustración 38. Distancia Punto referencia - Aeropuerto Portland

Minutes Delayed.Late Aircraft

calcular_distancia("$Statistics.MinutesDelayed.LateAircraft")

_id			Minutes_Delayed
Location	Airport	Distance	
{ type : "Point", coordinates : [-122.5979996, 45.58869934] }	Portland, OR: Portland International	2268,449 (2.3K)	19.649,2895 (19.6K)
{ type : "Point", coordinates : [-77.037697, 38.8521] }	Washington, DC: Ronald Reagan Washington National	1777,1718 (1.8K)	23.361,6447 (23.4K)
{ type : "Point", coordinates : [-80.29060363769531, 25.79319953918457] }	Miami, FL: Miami International	2078,5362 (2.1K)	25.171,4013 (25.2K)

Ilustración 39. Top 3 aeropuertos con menor media de minutos de demora por retraso de vuelos

Al igual que en el caso anterior, Portland ocupa nuevamente el primer puesto, en este caso con menor media de minutos de demora por retraso en vuelos, al que le sigue el aeropuerto de Washington D.C y Miami.

Si lo mostramos gráficamente mediante el *script pinta_coordenadas.py*:

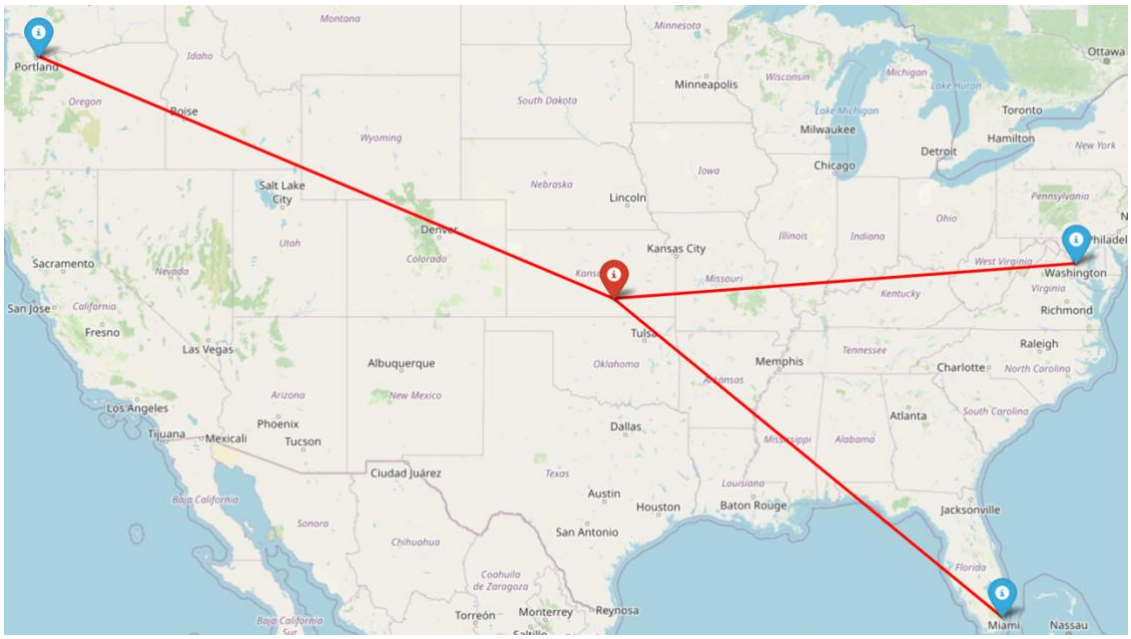

Ilustración 40. Mapa Minutes Delayed - Late Aircraft

Minutes Delayed.National Aviation System

calcular_distancia("$Statistics.MinutesDelayed.NationalAviationSystem")

_id			Minutes_Delayed
Location	Airport	Distance	
{ type : "Point", coordinates : [-122.5979996, 45.58869934] }	Portland, OR: Portland International	2268,449 (2.3K)	6875,9276 (6.9K)
{ type : "Point", coordinates : [-117.190002441, 32.7336006165] }	San Diego, CA: San Diego International	1881,7065 (1.9K)	11.352,9013 (11.4K)
{ type : "Point", coordinates : [-82.533203125, 27.975500106811523] }	Tampa, FL: Tampa International	1750,2239 (1.8K)	12.852,6711 (12.9K)

Ilustración 41. Top 3 aeropuertos con menor media de minutos de demora por el Sistema Nacional de Aviación

De nuevo en el podio, nos encontramos con el aeropuerto de Portland como el aeropuerto con menor media de minutos de demora debido a posibles fallas en el Sistema Nacional de Aviación estadounidense (conjunto de instrumentos y equipos de comunicación tanto civiles como militares para la correcta navegación aérea). Le sigue el aeropuerto de San Diego junto con, nuevamente, el aeropuerto de Tampa.

Si lo mostramos gráficamente mediante el *script pinta_coordenadas.py*:

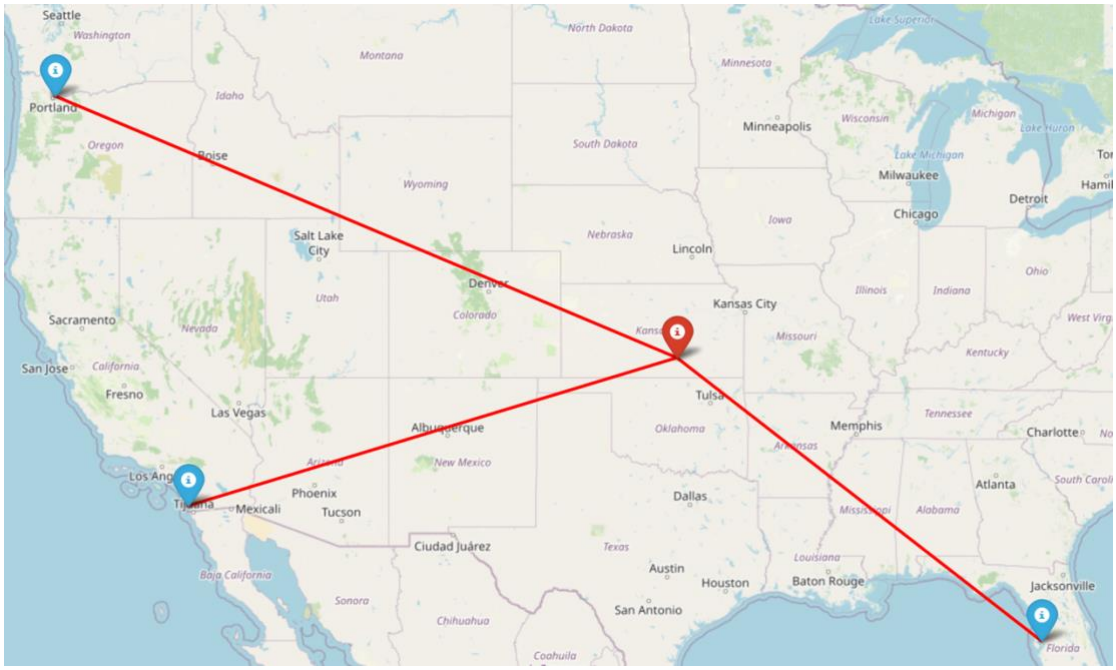

Ilustración 42. Mapa Minutes Delayed - National Aviation System

Minutes Delayed.Security

calcular_distancia("$Statistics.MinutesDelayed.Security")

_id			Minutes_Delayed
Location	Airport	Distance	
{ type : "Point", coordinates : [-73.872597, 40.777199] }	New York, NY: LaGuardia	2048,5909 (2.0K)	82,7829
{ type : "Point", coordinates : [-77.45580292, 38.94449997] }	Washington, DC: Washington Dulles International	1740,6566 (1.7K)	90,7697
{ type : "Point", coordinates : [-77.037697, 38.8521] }	Washington, DC: Ronald Reagan Washington National	1777,1718 (1.8K)	96,75

Ilustración 43. Top 3 aeropuertos con menor media de minutos de demora por Seguridad

Curiosamente, los aeropuertos con menor número medio de minutos de demora por Seguridad residen tanto en Nueva York, un aeropuerto cuya seguridad ha sido reforzada con el transcurso de los años desde los atentados del 11 de septiembre, hasta la misma capital del estado: Washington D.C, cuya seguridad no solo se rige a nivel civil sino además presidencial.

Si lo mostramos gráficamente mediante el *script pinta_coordenadas.py*:

Ilustración 44. Mapa Minutes Delayed – Security

Minutes Delayed.Weather

calcular_distancia("$Statistics.MinutesDelayed.Weather")

_id			Minutes_Delayed
Location	Airport	Distance	
{ type : "Point", coordinates : [-122.5979996, 45.58869934] }	Portland, OR: Portland International	2268,449 (2.3K)	1360,6974 (1.4K)
{ type : "Point", coordinates : [-117.190002441, 32.7336006165] }	San Diego, CA: San Diego International	1881,7065 (1.9K)	2438,5461 (2.4K)
{ type : "Point", coordinates : [-80.152702, 26.072599] }	Fort Lauderdale, FL: Fort Lauderdale-Hollywood International	2067,3728 (2.1K)	2546,1053 (2.5K)

Ilustración 45. Top 3 aeropuertos con menor media de minutos de demora por causas meteorológicas

Curiosamente, Portland repite como primera posición en esta clasificación, junto con ciudades como San Diego o Fort Lauderdale, ciudades cuyas condiciones climáticas no son tan extremas (en especial con las tormentas de nieve como ya mencionamos previamente), en especial en los estados del sur como California y Florida, por lo que el tiempo de demora por causas meteorológicas en estos aeropuertos es menor.

Si lo mostramos gráficamente mediante el *script pinta_coordenadas.py*:

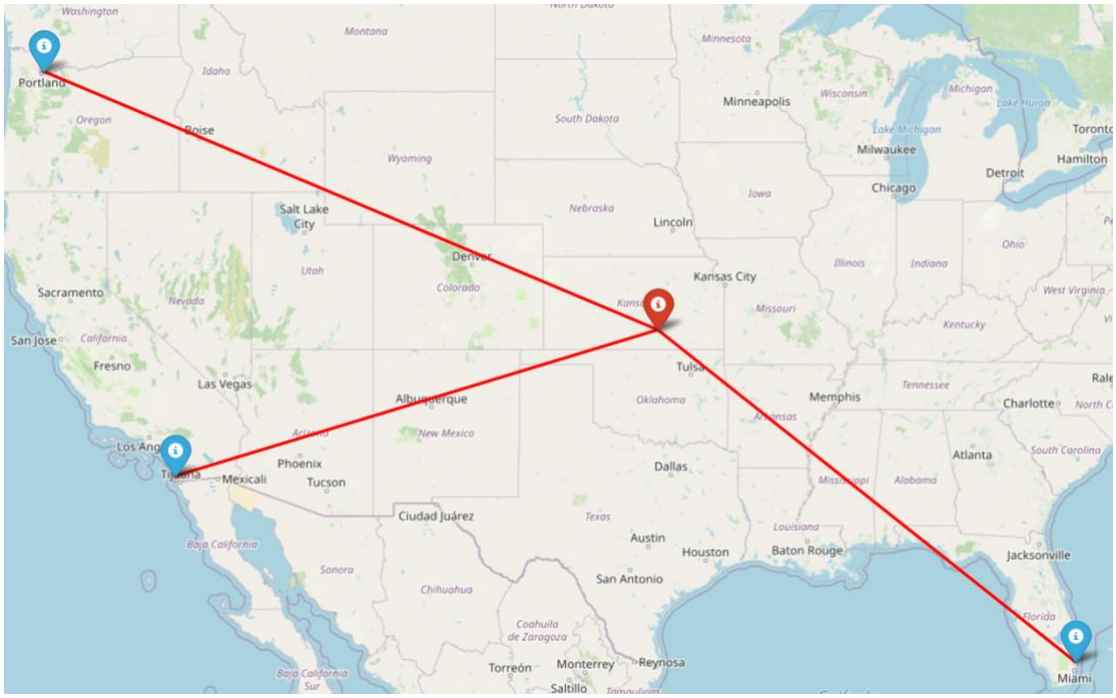

Ilustración 46. Minutes Delayed – Weather

5.2. CONSULTAR LOS AEROPUERTOS (A 500 KM O MENOS DE MI UBICACIÓN) CUYA PROPORCIÓN MINUTOS_DEMORADOS / VUELOS_DEMORADOS SEA IGUAL O MENOR A 50

Como última consulta, podemos calcular no solo la distancia entre los aeropuertos, sino que además me filtre aquellos cuya distancia al punto de referencia no supere los 500 km, mediante el campo *maxDistance* (en metros), incluido en el operador *$geoNear*. De forma adicional, una vez filtrados los aeropuertos más cercanos, mediante el operador *$match* filtraremos aquellos cuya proporción *Minutes Delayed.Total / Flights.Delayed* sea igual o inferior a 50 minutos / vuelos demorado.

Nota: como punto de referencia utilizaremos las coordenadas a la ciudad de Santa María, ubicada en California. [6]

```
var parametros = {
    near: {type: "Point", coordinates: [ -120.426935, 34.939985 ] },
    distanceField: "dist.calculated",
    key: "Position",
    maxDistance: 500000,
    distanceMultiplier: 0.001,
    includeLocs: "dist.location",
    spherical: true
  }
var fase1 = {$geoNear: parametros}
var group = {_id: {"Location": "$dist.location", "Airport": "$Airport.Name", "Distance": "$dist.calculated"}, "minutos": {$sum: "$Statistics.MinutesDelayed.Total"}, "vuelos": {$sum: "$Statistics.Flights.Delayed"}}
var fase2 = {$group: group}
var project = {"_id": 1, "proporcion_minutos_vuelos": {$divide: ["$minutos", "$vuelos"]}}
var fase3 = {$project: project}
var cond = {$expr: {$lte: ["$proporcion_minutos_vuelos", 50]}}
var fase4 = {$match: cond}
```

	_id			proporcion_minutos_vuelos
	Location	Airport	Distance	
1	{ type : "Point", coordinates : [-117.190002441, 32.7336006165] }	San Diego, CA: San Diego International	387,1345	46,9272
2	{ type : "Point", coordinates : [-115.1520004, 36.08010101] }	Las Vegas, NV: McCarran International	494,4768	49,1274

Ilustración 47. Aeropuertos más cercanos con proporción menor o igual a 50

[6] https://es.wikipedia.org/wiki/Santa_Mar%C3%ADa_(California)

Como podemos observar, tanto el aeropuerto de Las Vegas como el aeropuerto de San Diego satisfacen ambas condiciones, con una proporción de 49 y 46 minutos de demora por cada vuelo demorado, respectivamente.

Si lo mostramos gráficamente mediante el *script pinta_coordenadas.py*:

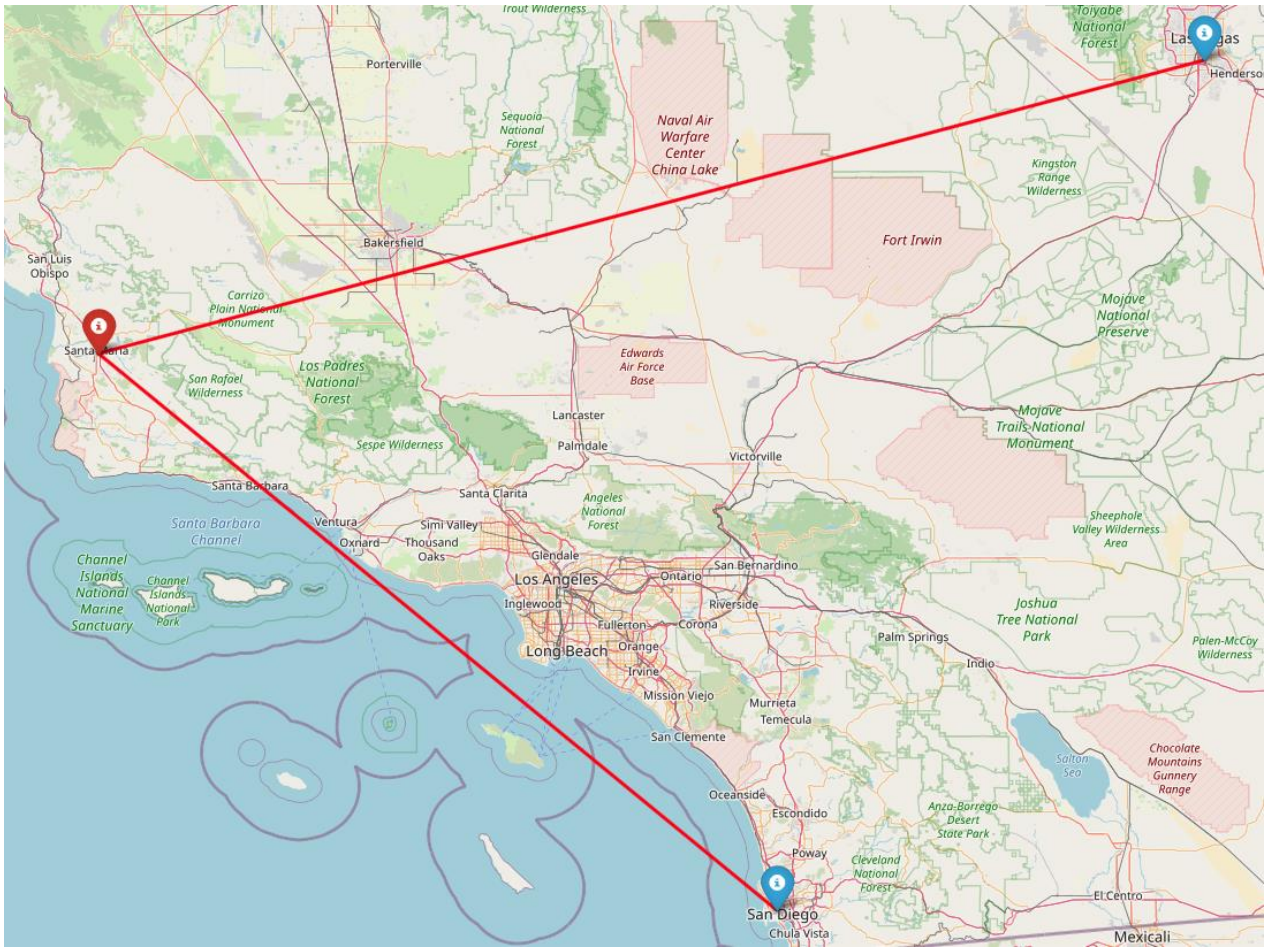

Ilustración 48. Aeropuertos más cercanos con menor proporción

6. CONCLUSIONES

A continuación, se enumeran las principales conclusiones obtenidas a partir de la información recabada del *dataset*:

1. Con el transcurso de los años, en especial entre los años 2003 y 2007, **el número de vuelos en Estados Unidos experimentó un aumento gradual en cuanto al número de pasajeros, aumento que se vio reducido tras la crisis del año 2008**, momento a partir del cual el número de vuelos comenzó a descender, experimentando subidas y bajadas entre diferentes años, en especial durante el comienzo de la crisis.
2. En relación con las estadísticas de vuelo, tras los análisis realizados en el apartado 4.3, **los aeropuertos de Chicago, Atlanta y Dallas encabezan la lista con el mayor número de llegadas a tiempo, pero siendo además aquellos aeropuertos donde más vuelos se han cancelado o demorado**, dado que al ser los más concurridos, supone también un mayor número de incidencias. Cabe

destacar con especial atención el número de vuelos demorados por causas climatológicas, dado que estos aeropuertos están situados en la zona norte del país, siendo afectados por las tormentas de nieve que asolan el país durante los meses de invierno.

3. En relación con el apartado anterior, en el apartado 4.4 hemos podido comprobar como **durante los meses de diciembre, enero y febrero, el número medio de cancelaciones aumenta considerablemente, en especial en los aeropuertos situados al norte del país, incluyendo Chicago, Atlanta y Dallas**, previamente mencionados, meses en los cuales las intensas nevadas obligan a demorar o cancelar multitud de vuelos, en combinación con el periodo navideño, época de mayor movimiento en los aeropuertos.
4. Sin embargo, una de las peores situaciones en un aeropuerto no solo es el momento en el que un vuelo es cancelado, sino incluso cuando un vuelo es demorado, en especial cuando la demora se prolonga durante media hora o incluso horas. En la consulta 4.5 hemos podido comprobar cómo **el aeropuerto de Nueva Jersey encabeza el listado de aeropuertos con mayor proporción minutos_demora / vuelos_demorado**, posición en la que, curiosamente, no están incluidos los aeropuertos de Chicago, Atlanta o Dallas, lo que significaría que el número de demoras en estos aeropuertos es mayor, pero su duración (en minutos) es menor, situación contraria a la que sucede en Nueva Jersey, donde la media de demora es de algo más de una hora (68 minutos) por cada vuelo demorado. Por el contrario, en la última posición del *top* nos encontramos con el aeropuerto de San Diego, cuya proporción apenas supera los 45 minutos.
5. Dejando a un lado las estadísticas de vuelos, los datos de compañías aéreas resulta ser un campo fundamental para analizar. En la consulta 4.6 hemos podido comprobar cómo **el número medio de compañías aéreas, de igual modo que ocurría con el número de vuelos, creció entre los años 2003 y 2007, aunque tras el estallido de la crisis el número medio comenzó a descender**, pasando de una media de 12 compañías áereas en el año 2003, 14 compañías en el año 2007, hasta 9 a comienzos del 2016. Esto último nos hace pensar qué compañías han ido desapareciendo entre años consecutivos, cuestión que nos resuelve la consulta 4.7, donde pudimos observar cómo, mayoritariamente, compañía de vuelos regionales o de bajo coste, como *Aloha Airlines*, *America West Airlines* o *ATA Airlines*, han desaparecido (con especial atención a la crisis del año 2008), siendo absorbidas en su mayoría por las grandes compañías de vuelo estadounidenses (consulta 4.8) que conocemos y que se mantienen hasta el día de hoy: *American Airlines*, *Hawaian Airlines*, *Jet Blue Airways* o *United Airlines*, entre otras, lo que les han permitido no sólo un mayor número de pasajeros, sino además una mayor infraestructura para sus operaciones.
6. Junto con la utilización de coordenadas para el cálculo de distancia entre aeropuertos, hemos podido analizar los valores del campo *Minutes Delayed*, en especial aquellos aeropuertos que presentan el menor valor medio de minutos de demora (apartado 5.1). Estudiando cada uno de sus campos (*Carrier*, *Late Aircraft*, *National Aviation System*, *Security* y *Weather*), podemos concluir que **el aeropuerto Internacional de Portland presenta el menor valor medio en prácticamente todos los campos anteriores**, salvo en el campo *Security*,

encabezados por los aeropuertos de Nueva York y Washington, aeropuertos cuya seguridad se vio fuertemente reforzada tras los atentados del 11 de septiembre.

En relación con los *datasets*, quisiera remarcar que no se han dispuesto de estadísticas de vuelos de años posteriores a 2016. En caso de haberlos tenido, veríamos probablemente un aumento gradual en el número de vuelos, aumento que se hubiera visto interrumpido en el año 2020, con la pandemia del COVID-19, momento en el cual se ha experimentado una gran caída en el número de viajes.