DATASET

SUCESOS AERONÁUTICA ARGENTINA

BASES DE DATOS NOSQL

DAYRON ONEL RODRIGUEZ DIAZ

MONGO DB

1. Introducción

En un entorno cada vez más orientado a los datos, la capacidad de estructurar, almacenar y acceder eficientemente a la información se ha convertido en un factor clave para el éxito de cualquier sistema informático. Las bases de datos no solo permiten organizar grandes volúmenes de datos, sino que también facilitan la toma de decisiones, la automatización de procesos y la mejora continua en diversos ámbitos como la industria, el comercio, la salud y el transporte.

Este trabajo presenta una base de datos NoSQL orientada al almacenamiento y análisis de información sobre accidentes aeronáuticos ocurridos en Argentina. A diferencia de los modelos relacionales tradicionales, el enfoque NoSQL permite manejar datos semi-estructurados y no estructurados, facilitando la integración de registros históricos, informes técnicos, coordenadas geográficas, metadatos operacionales y testimonios en formatos variados.

El conjunto de datos utilizado ofrece un alto valor informativo, ya que permite estudiar en profundidad los accidentes aéreos, un tema de gran relevancia para la seguridad y eficiencia del transporte aéreo. El análisis de estos eventos resulta fundamental para identificar patrones, establecer medidas preventivas y contribuir a la mejora continua de uno de los medios de transporte más utilizados a nivel global.

2. Análisis de la Base de Datos

2.1 Kaggle

Kaggle es una plataforma web que reúne la comunidad Data Science más grande del mundo, con más de 536 mil miembros activos en 194 países, recibe más de 150 mil publicaciones por mes, que brindan todas las herramientas y recursos más importantes para progresar al máximo en Data Science. Kaggle, al igual que DataScientest, tiene una interfaz Jupyter Notebooks personalizable y sin configuración. Permite acceder de manera gratuita a GPUs y a una gran cantidad de datos y códigos publicados por la comunidad. En Kaggle, encontrarás los códigos y datos que necesitas para realizar tus proyectos data science. Hay más de 50 mil conjuntos de datos públicos y 400 mil notas públicas disponibles para todo el mundo.

La base de datos "Sucesos Aeronáutica Argentina" se obtuvo de Kaggle, donde aparte de obtener el archivo .csv, también se pueden apreciar análisis gráficos y de otra naturaleza acerca de las bases de datos publicadas.

2.2 MongoDB

Una vez descargado el .csv desde la plataforma, se procede a la importación de los datos en el software MongoDB.

Según la web del sitio, MongoDB se define como:

"MongoDB es una base de datos de documentos que ofrece una gran escalabilidad y flexibilidad, y un modelo de consultas e indexación avanzado."

Por lo que se puede llegar a la conclusión de que es un sistema de gestión de bases de datos de tipo NoSQL, lo que se traduce a una base de datos no relacional que guarda la información en documentos flexibles en vez de tablas y filas como otros sistemas.

2.2.1 Importación de los datos

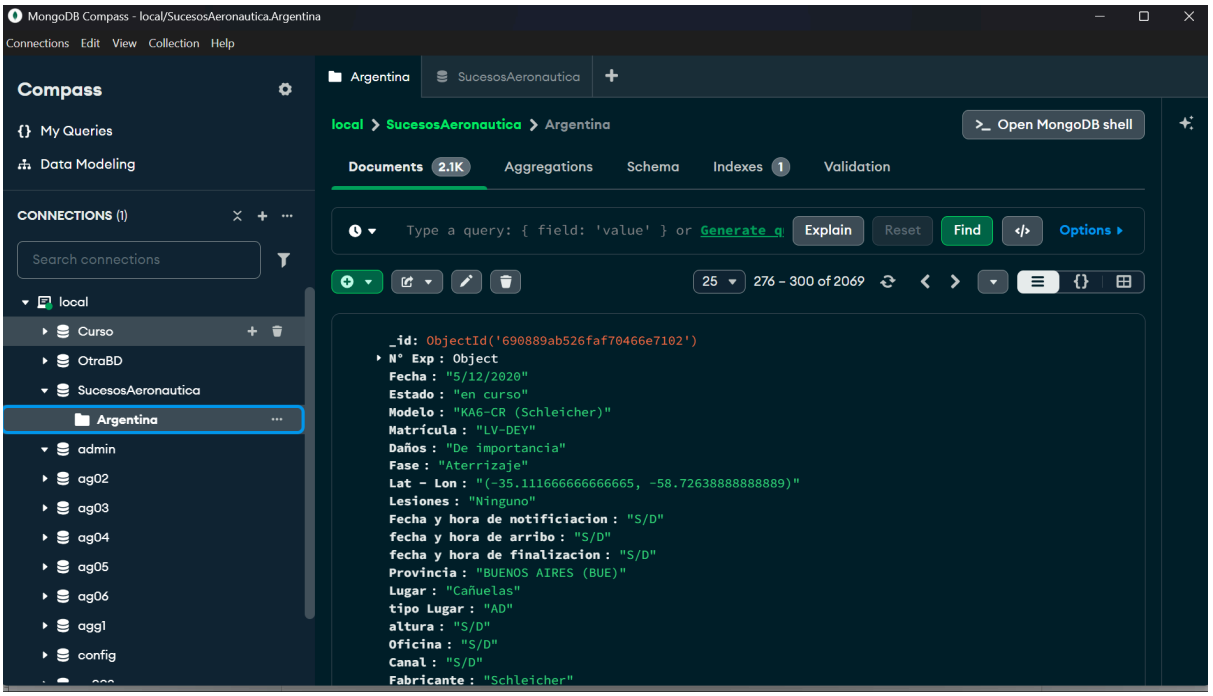

Imagen 2.1 Entorno MongoBD

Como se puede apreciar en la Imagen 2.1, se ha creado la base de datos “SucesosAeronáutica” y se le agrega la colección Argentina. Se modela de esta manera porque se tiene pensado en un futuro ampliar la misma, agregando nuevas colecciones con los conjuntos de datos de otros países.

Centrando la atención, en la colección Argentina, es una colección bastante amplia de datos, teniendo un total de 2.1k documentos:

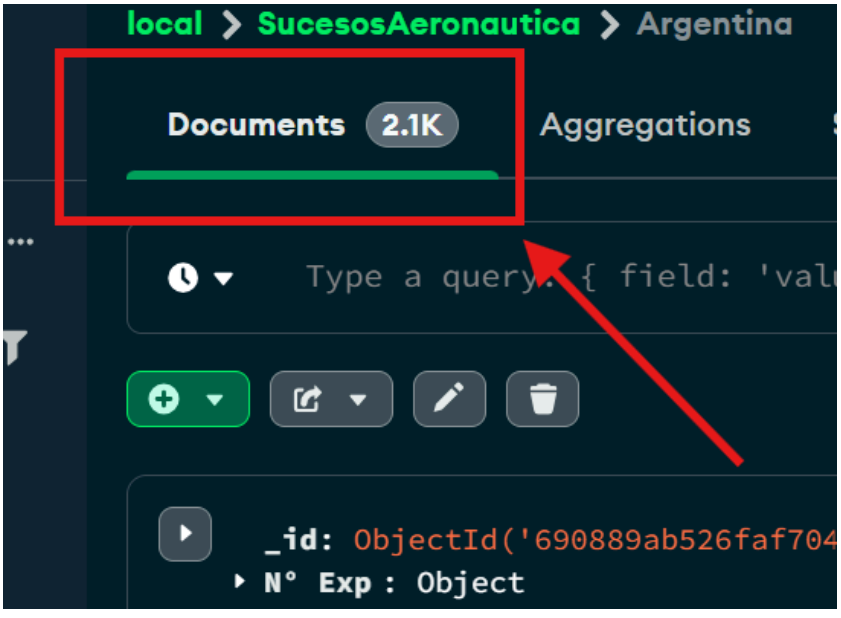

Imagen 2.2 Documentos de la colección

2.3 Estructura de los documentos

Imagen 2.3 Estructura de un documento

Partiendo de la imagen anterior, se procede a analizar la estructura de un documento o registro:

- **_id:** identificador único de 12 bytes que MongoDB asigna por defecto al campo _id de cada documento. Esto garantiza unicidad incluso en entornos distribuidos. (ObjectId).
- **N° Exp:** Número de expediente (string).
- **Fecha**: Fecha del suceso en formato DD/MM/YYYY (string).
- **Estado:** Estado del expediente, si está en curso o cerrado (string).
- **Modelo:** Modelo de la aeronave (string).
- **Matrícula:** Identificador de la aeronave (string).
- **Daños:** Nivel de daño reportado (string).
- **Fase:** Fase del vuelo en la que ocurrió el evento (string).
- **Lat-Lon:** Coordenadas del accidente (string).
- **Lesiones:** Lesiones reportadas (string).
- **Fecha y hora de notificación:** Fecha/hora de notificación (string).
- **Fecha y hora de arribo:** Fecha/hora de arribo al lugar del suceso (string).
- **Fecha y hora de finalización:** Fecha/hora de finalización del expediente (string).
- **Provincia:** Provincia donde ocurrió el evento (string).
- **Lugar:** Ciudad o localidad del suceso (string).
- **Tipo Lugar:** Tipo de instalación (ejemplo: Aeropuerto)(string).
- **Altura:** Altura de la aeronave al momento del evento (string).
- **Oficina:** Oficina responsable (string).
- **Canal:** Canal de comunicación (string).
- **Fabricante:** Fabricante de la aeronave (string).

2.4 Preprocesamiento y validación

Antes de realizar consultas y análisis sobre el dataset, es necesario revisar la calidad y consistencia de los datos. A continuación, se detallan las principales observaciones y acciones tomadas:

2.4.1 Altura, Oficina y Canal

Se identificaron múltiples campos con el valor “S/D” (sin datos), lo cual representa una forma no estándar de codificar valores faltantes. Para facilitar el análisis, estos valores fueron transformados a null.

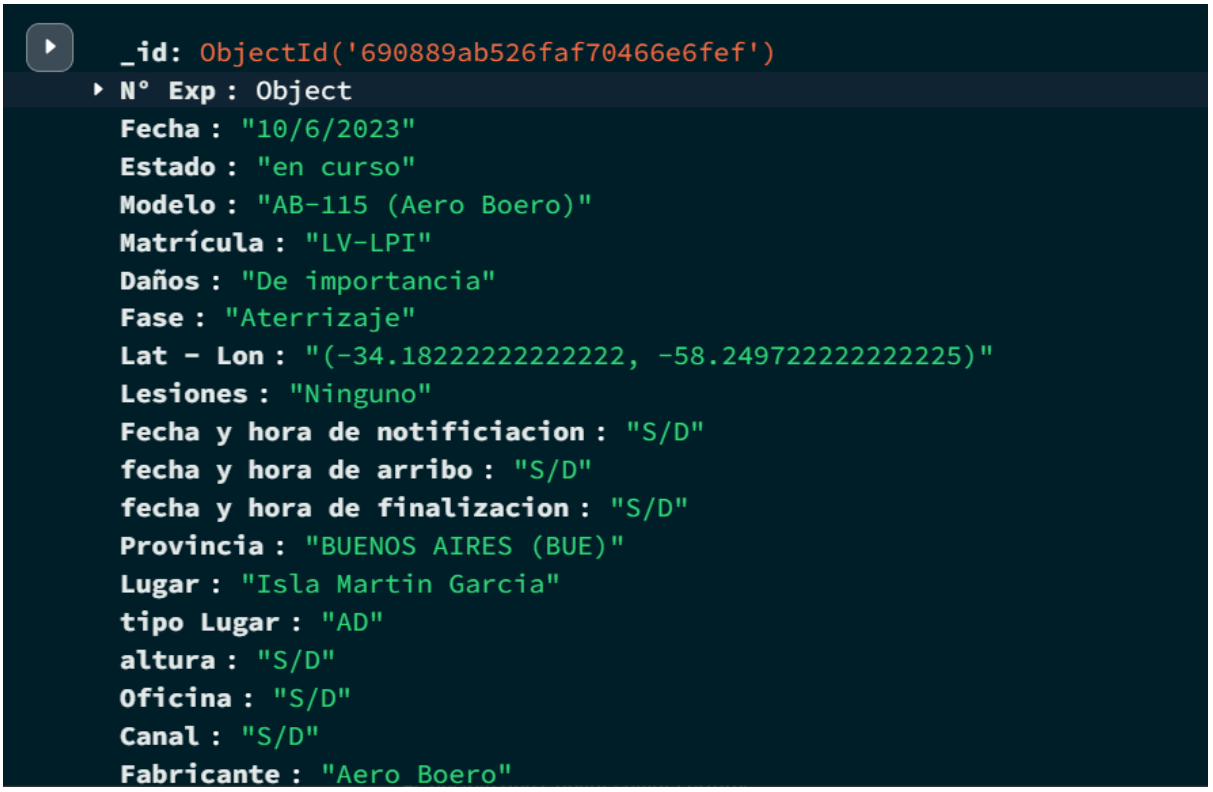

Imagen 2.4 Documento antes de ser corregido

Para lograr esto, se accede a la consola y a través de un comando, se corrigen estos valores como se muestra a continuación:

(Actualización)

```
db.Argentina.updateMany({"altura":"S/D"},{$set: {"altura":null}} )
```

```
> db.Argentina.updateMany({"altura":"S/D"},{$set: {"altura":null}} )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 2069,
    modifiedCount: 2069,
    upsertedCount: 0
  }
SucesosAeronautica>
```

Imagen 2.5 Comando para lograr la modificación del valor de "Altura"

Observando la imagen anterior, se muestra como la modificación se realiza con éxito, arrojando un total de 2069 documentos actualizados. Si se comprueba con cualquier documento, luego de refrescar la base de datos, se puede verificar el cambio:

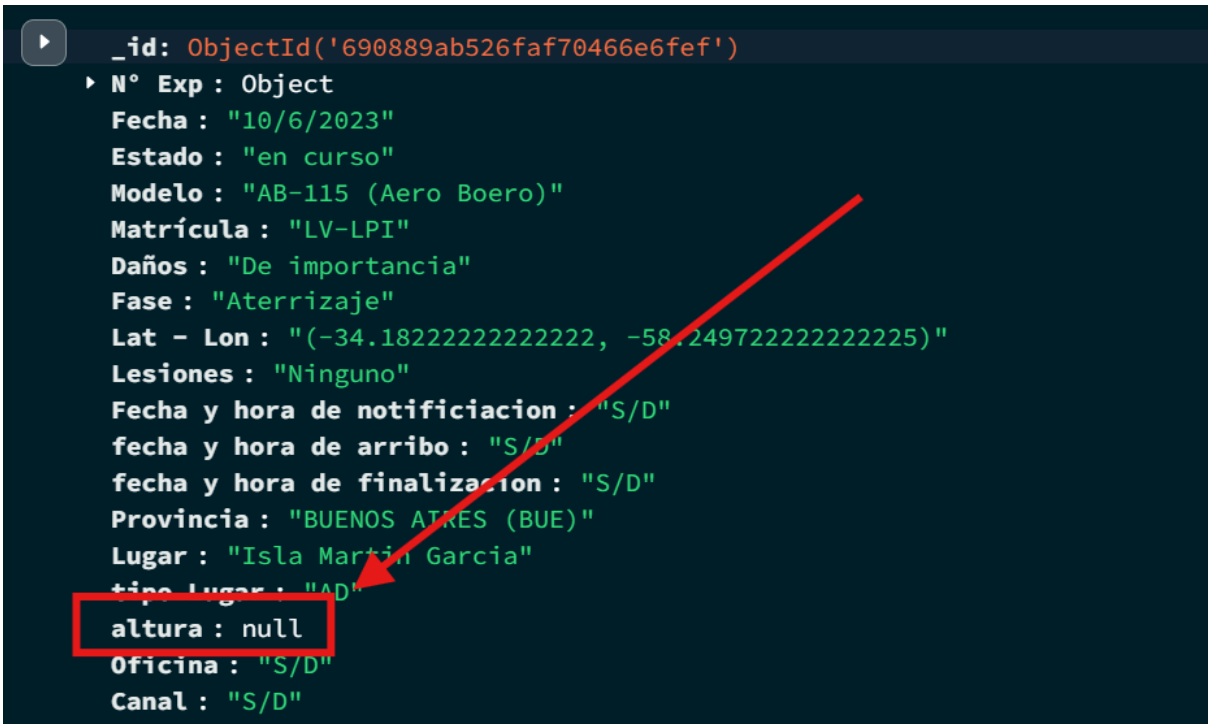

Imagen 2.6 "Altura modificada"

Este mismo cambio se aplica para los otros campos que también contienen el valor de "S/D", que serían Oficina y Canal. Luego de este análisis, se comentan a los comandos utilizados y los resultados arrojados:

(Actualización)

db.Argentina.updateMany({"Oficina":"S/D"},{$set: {"Oficina":null}})

```
> db.Argentina.updateMany({"Oficina":"S/D"},{$set: {"Oficina":null}} )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 2069,
    modifiedCount: 2069,
    upsertedCount: 0
  }
```

Imagen 2.7 Comando para lograr la modificación del valor de "Oficina"

(Actualización)

db.Argentina.updateMany({"Canal":"S/D"},{$set: {"Canal":null}})

```
> db.Argentina.updateMany({"Canal":"S/D"},{$set: {"Canal":null}} )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 0,
    modifiedCount: 0,
    upsertedCount: 0
  }
```

Imagen 2.8 Comando para lograr la modificación del valor de "Canal"

Una vez realizado esto, se refresca la base de datos y así se visualiza un documento:

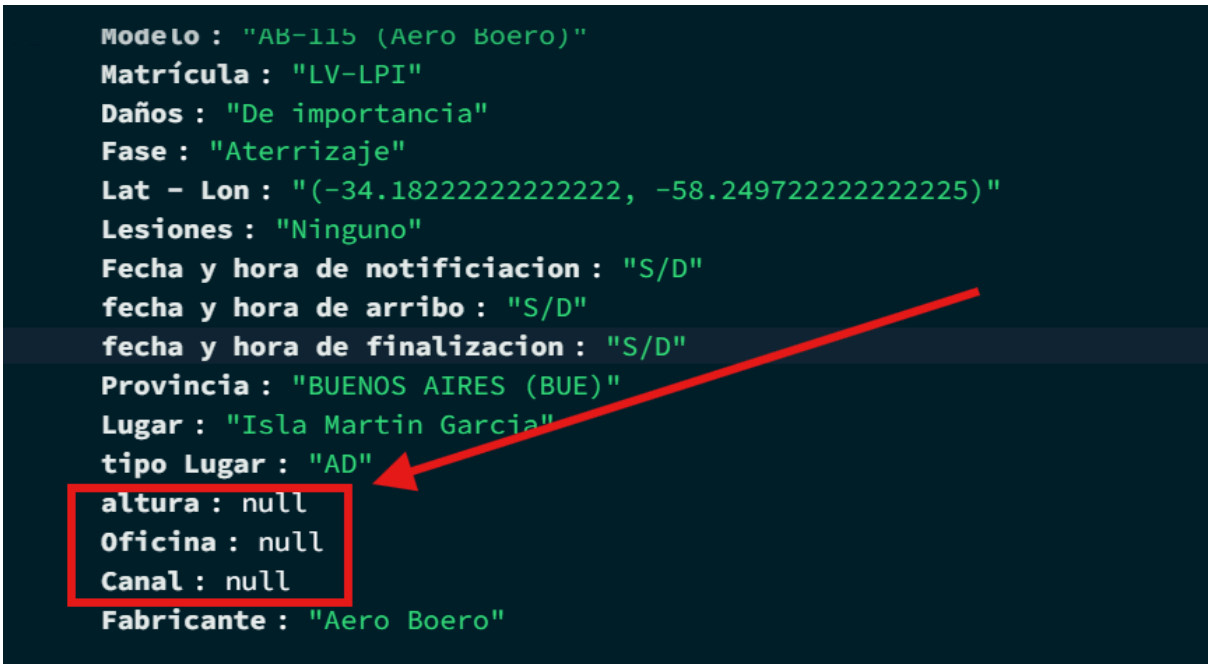

Imagen 2.9 Visualización de un documento una vez modificado "Altura","Oficina" y "Canal"

2.4.2 'N° Exp'

En todos los documentos de la base de datos, el campo 'N° Exp' se trataba como un subdocumento con clave vacía y no como un string directo.

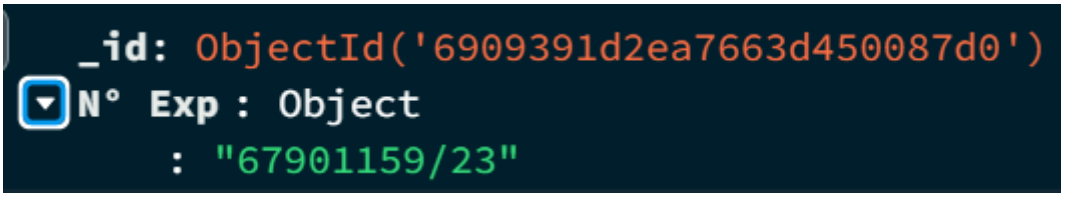

Imagen 2.10 Documento con 'N° Exp' como subdocumento con clave vacía

Este diseño causa problemas al consultar, filtrar o indexar 'N° Exp'. La idea que se busca es lograr que el número de expediente sea una cadena sencilla, para así poder trabajar con este campo de una manera más práctica. Para arreglar este problema se utiliza el siguiente comando:

(Actualización con transformación estructural)

```
db.Argentina.find().forEach(doc => {
  if (doc["N° Exp"] && doc["N° Exp"][""] !== undefined) {
    db.Argentina.updateOne(
      { _id: doc._id },
      { $set: { "N° Exp": doc["N° Exp"][""] } }
    );
  }
```

```
});
```

Básicamente con este comando lo que se busca es que se recorran todos los elementos de la base de datos, si hay un documento, que su número de expediente existe y a su vez, la subclave vacía tiene un valor definido, entonces que reemplace el objeto completo por el valor string hallado.

Una vez aplicados los cambios, así quedan los 'N° Exp' de cada documento:

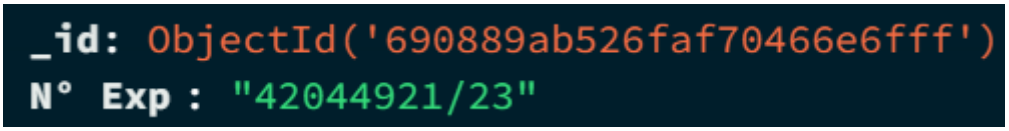

Imagen 2.10 Documento con 'N° Exp' como string

Otro punto que importante sería el nombre del campo, puesto que el carácter “ ° ”, complejiza un poco a la hora de filtrar o consultar para el analista, por lo que se procede a cambiar el nombre del campo:

(Actualización)

```
db.Argentina.updateMany({}, { $rename: { "N° Exp": "NroExp" } })
```

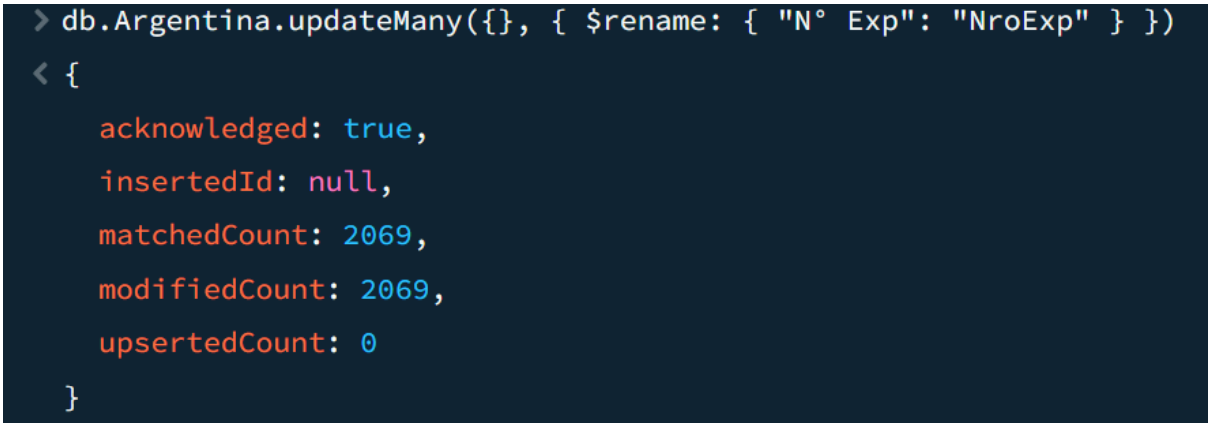

Imagen 2.11 Comando para lograr la modificación del nombre del campo

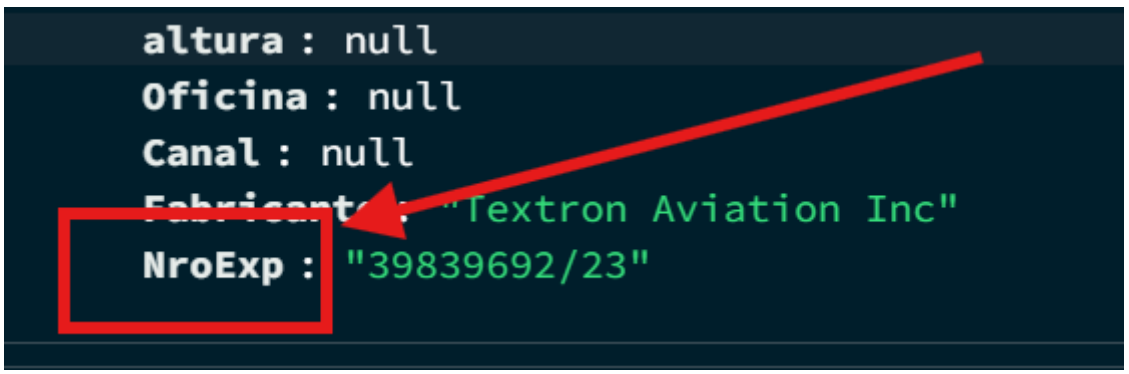

Imagen 2.12 Visualización de un documento una vez modificado el nombre del NroExp

2.4.3 Fecha

Si se observa, las fechas del documento son un string, lo cual no conviene a la hora de tratar los datos tipo fecha. Por lo que a través de un pipeline de agregación, se convierte este string en fecha:

(Pipeline de agregación - Proyección y Actualización)

```
[
  {
    $addFields: {
      Date: {
        $dateFromString: {
          dateString: "$Fecha",
          format: "%d/%m/%Y"
        }
      }
    }
  },

  {$merge: {
    into: "Argentina",
    whenMatched: 'merge',
    whenNotMatched: 'discard'
  }}
]
```

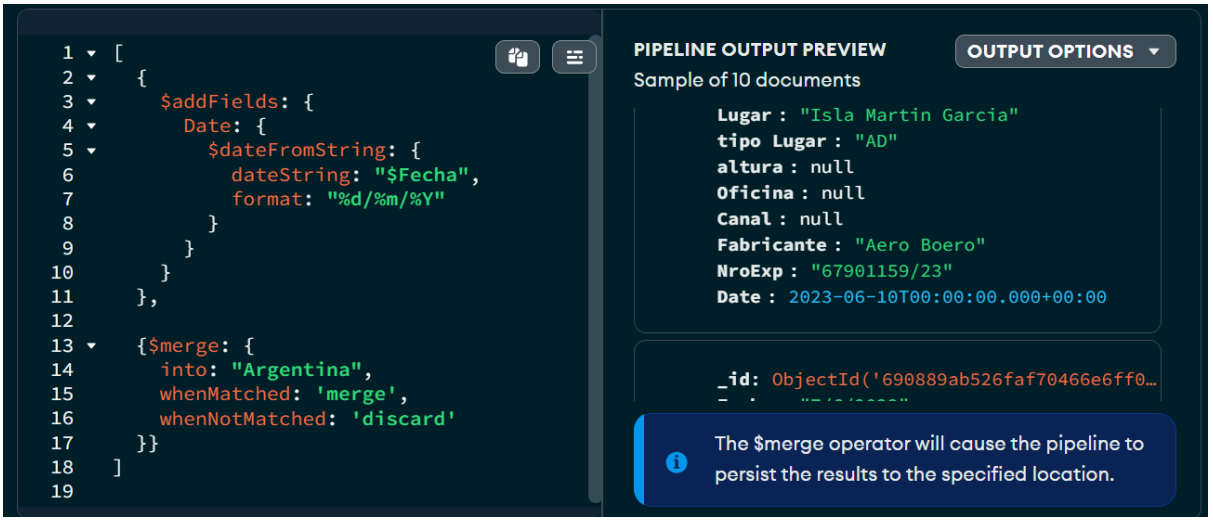

Imagen 2.13 Cambio de fecha en string a tipo Date

Una vez se ejecuta el código y se actualiza la base de datos, queda así el documento:

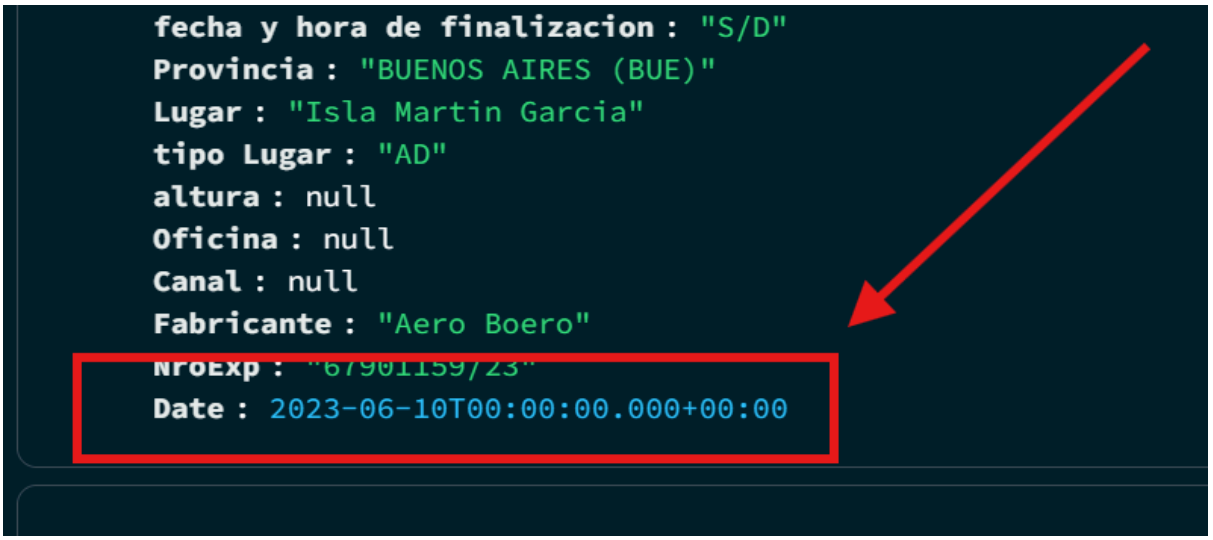

Imagen 2.14 Date como nuevo campo

Ahora se procede a eliminar el antiguo campo “Fecha”:

(Actualización)

```
db.Argentina.updateMany({},{$unset:{"Fecha":""}})
```

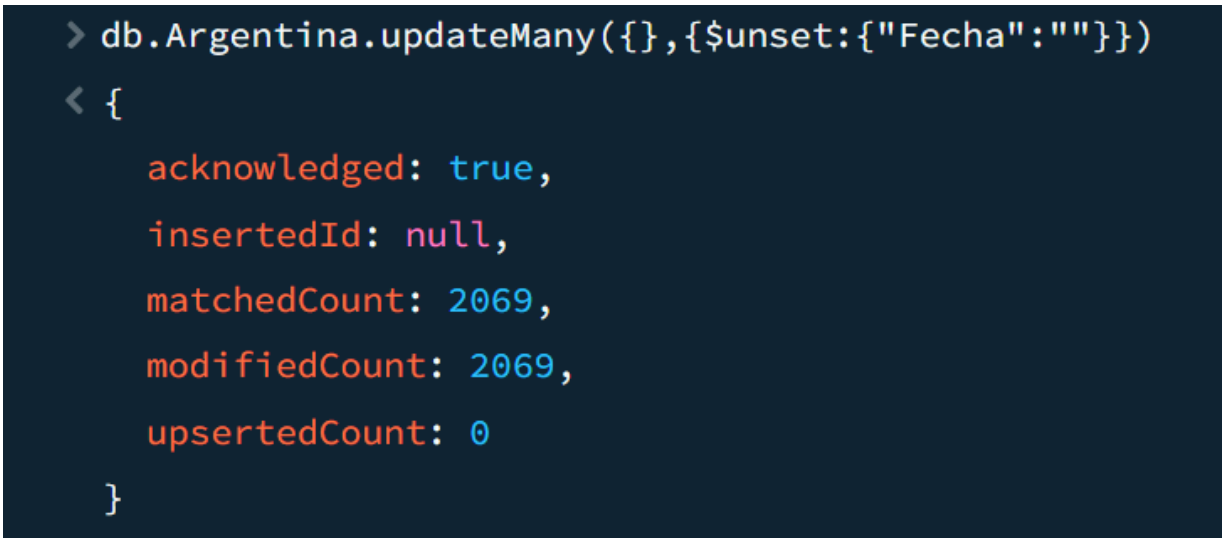

Imagen 2.15 Comando para eliminar el campo Fecha

2.4.5 Lat - Lon

En cada documento hay un campo denominado “Lat - Lon”, el cual se importa como un string, pero claramente se evidencia que es un objeto geoespacial. Para hacer esta normalización se detallan los pasos a continuación:

(Pipeline de agregación - Proyección y Actualización)

```
[{
  $addFields: {
    ubicacion: {
      $let: {
        vars: {
          partes: {
            $map: {
              input: {
                $split: [
                  {
                    $replaceAll: {
                      input: {
                        $replaceAll: {
                          input: "$Lat - Lon",
                          find: "(",
                          replacement: ""
                        }
                      },
                      find: ")",
                      replacement: ""
                    }
                  },
                  ","
                ]
              },
              as: "coord",
              in: {
                $convert: {
                  input: {
                    $trim: { input: "$$coord" }
                  },
                  to: "double",
                  onError: null,
                  onNull: null
                }
              }
            }
          }
        }
      }
    },
```

```
        in: {
          type: "Point",
          coordinates: [
            { $arrayElemAt: ["$$partes", 1] }, // longitud
            { $arrayElemAt: ["$$partes", 0] } // latitud
          ]
        }
      }
    }
  }
},
{
  $merge: {
    into: "Argentina",
    whenMatched: "merge",
    whenNotMatched: "discard"
  }
}]
```

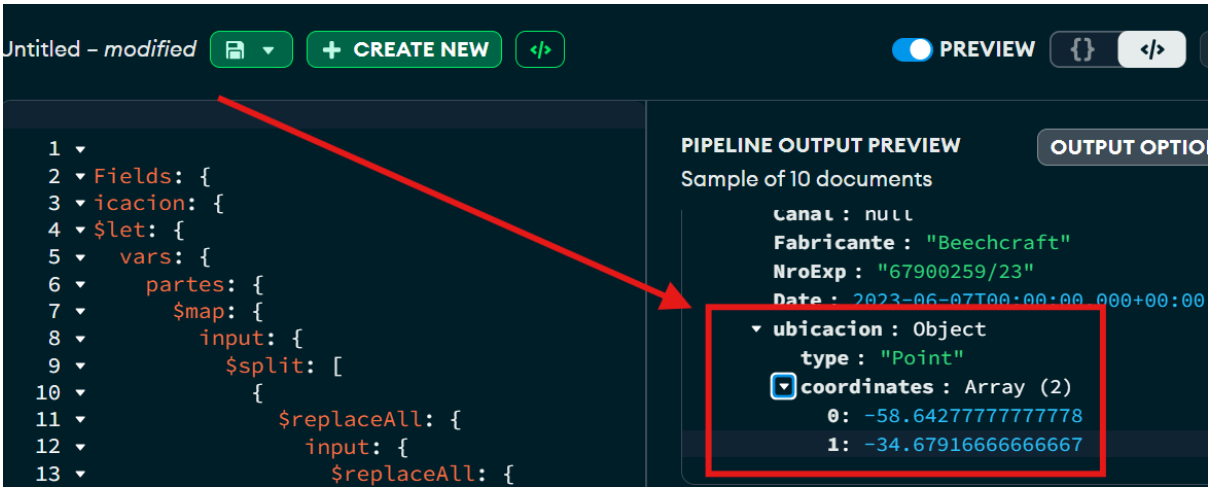

Imagen 2.16 Ubicación como objeto geoespacial

Este código lo que hace en un principio es agregar un campo ubicación, para esto se crea una variable temporal “partes”, la cual va almacenar los valores de las coordenadas. En un primer momento, para lograr la transformación a un campo geoespacial, el string contiene “()”, lo cual no va con este formato, por lo que se utiliza el operador $replaceAll para eliminar los paréntesis. Posteriormente, se utiliza $split para dividir los valores del string en un array de dos elementos(latitud y longitud). Una vez logrado esto, se recorre el array con $map y en coordinación con $convert se convierten estas cadenas de texto en datos de tipo double, los cuales para eliminar los espacios se optimizan con $trim. Un objeto geoJSON por defecto su estructura debe ser primero la longitud y luego la latitud, si se observa se llega a la conclusión que están en sentido inverso, por lo que hay que ajustar la posición de los

elementos del array. Para esto se utiliza $arrayElemAt y se encapsulan en un objeto con type:Point.

Por último, se utiliza $merge para integrar estos nuevos cambios a la base de datos. Una vez aplicados los cambios, así se visualiza un documento:

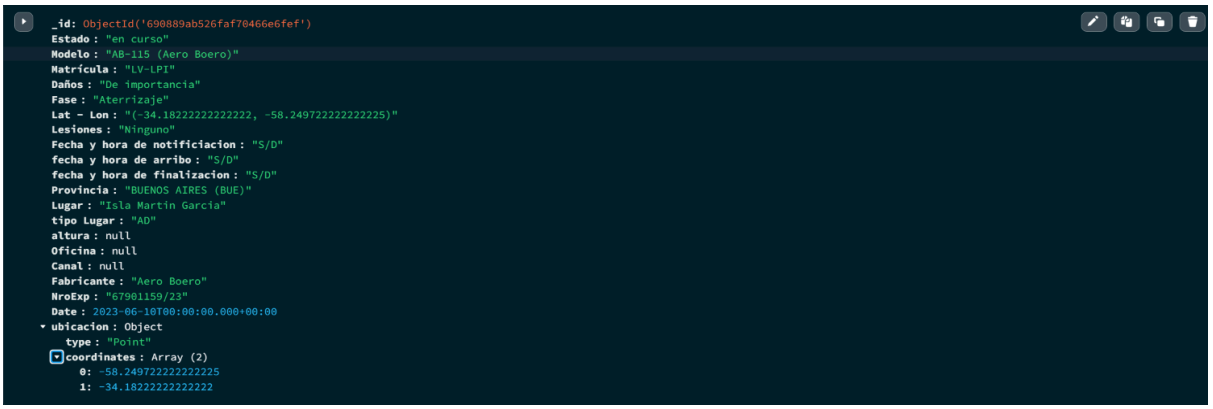

Imagen 2.17 Documento con ubicación

Una vez logrado esto, se procede a eliminar el campo “Lat - Lon”:

(Pipeline de agregación - Actualización)

```
[{$unset: 'Lat - Lon'}]
```

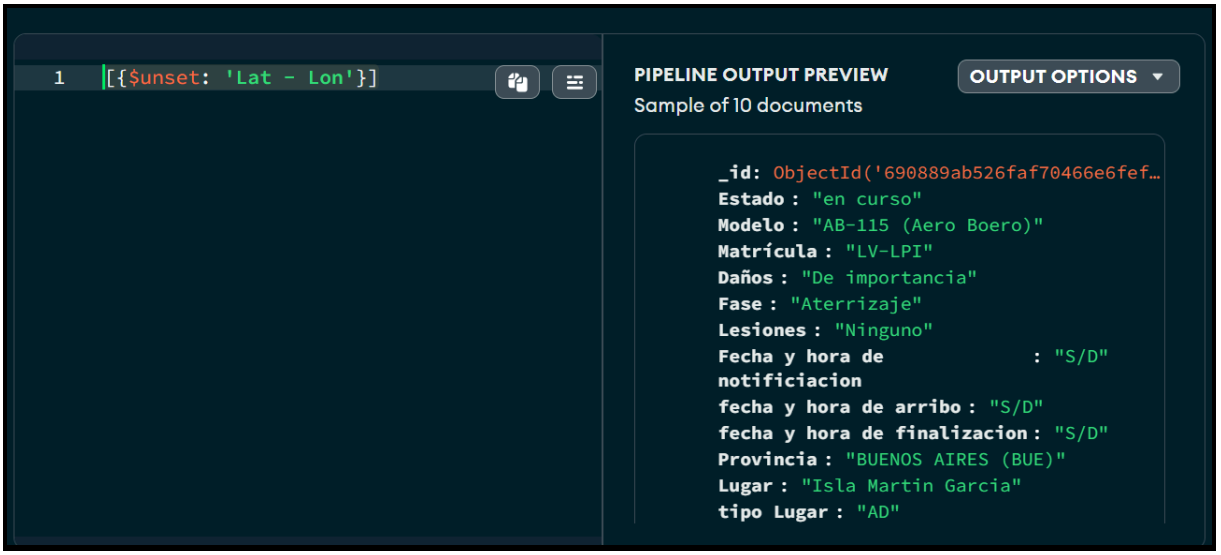

Imagen 2.18 Pipeline de agregación para eliminar el campo “Lat - Lon”

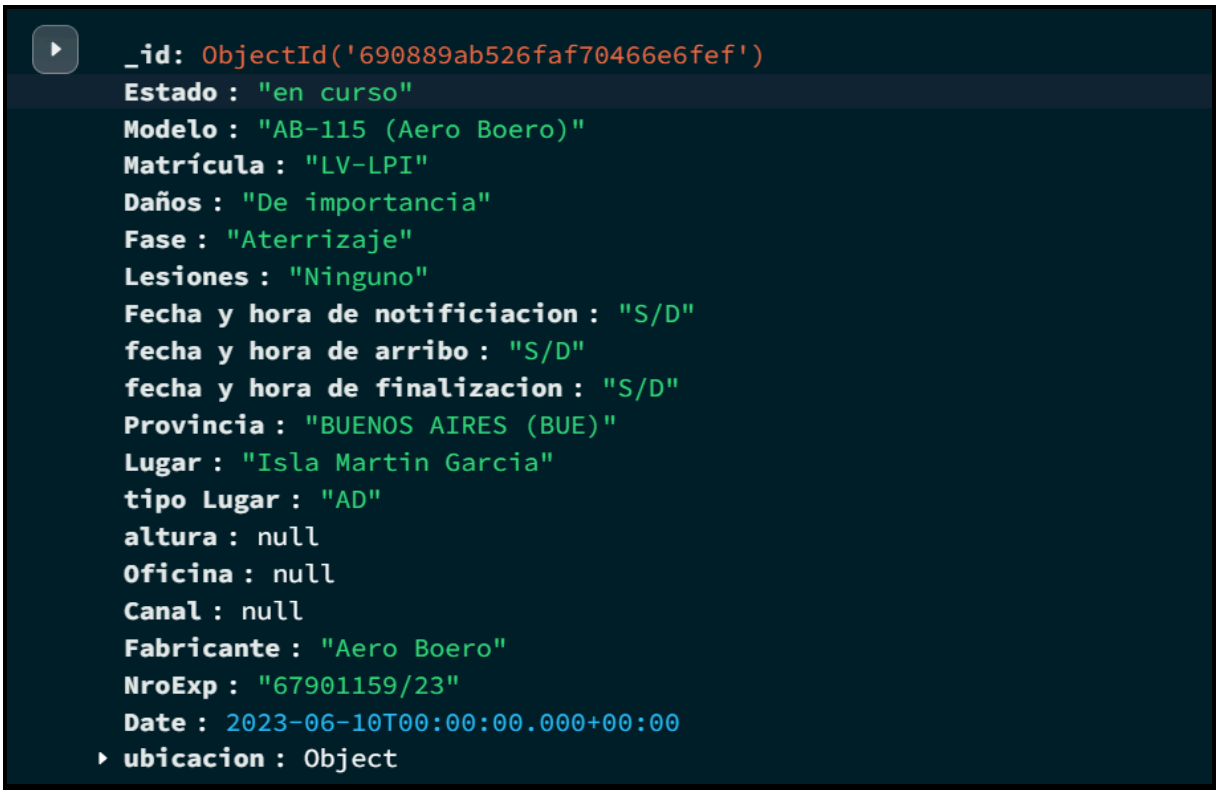

Imagen 2.19 Documento luego de eliminar el campo "Lat - Lon"

2.4.6 "Fecha y hora de notificación","fecha y hora de arribo","fecha y hora de finalización"

A estos campos se les dará el mismo tratamiento que a Altura, Oficina y Canal. Básicamente se están agregando como un string "S/D" (sin/datos), por lo que se procede a convertirlos en nulls.

(Actualización)

```
db.Argentina.updateMany({"Fecha y hora de notificiacion":"S/D"},{$set:{"Fecha y hora de notificiacion":null}})
```

Imagen 2.20 Comando para lograr la modificación del valor de "Fecha y hora de notificación"

(Actualización)

```
db.Argentina.updateMany({"fecha y hora de arribo":"S/D"},{$set:{"fecha y hora de arribo":null}})
```

```
> db.Argentina.updateMany({"fecha y hora de arribo":"S/D"},{$set:{"fecha y hora de arribo":null}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 2069,
    modifiedCount: 2069,
    upsertedCount: 0
  }
SucesosAeronautica>
```

Imagen 2.21 Comando para lograr la modificación del valor de "Fecha y hora de arribo"

(Actualización)

```
db.Argentina.updateMany({"fecha y hora de finalizacion":"S/D"},{$set:{"fecha y hora de finalizacion":null}})
```

```
> db.Argentina.updateMany({"fecha y hora de finalizacion":"S/D"},{$set:{"fecha y hora de finalizacio
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 2069,
    modifiedCount: 2069,
    upsertedCount: 0
  }
SucesosAeronautica>
```

Imagen 2.22 Comando para lograr la modificación del valor de "Fecha y hora de finalización"

Una vez realizados todos estos cambios, así se visualiza un documento de la base de datos:

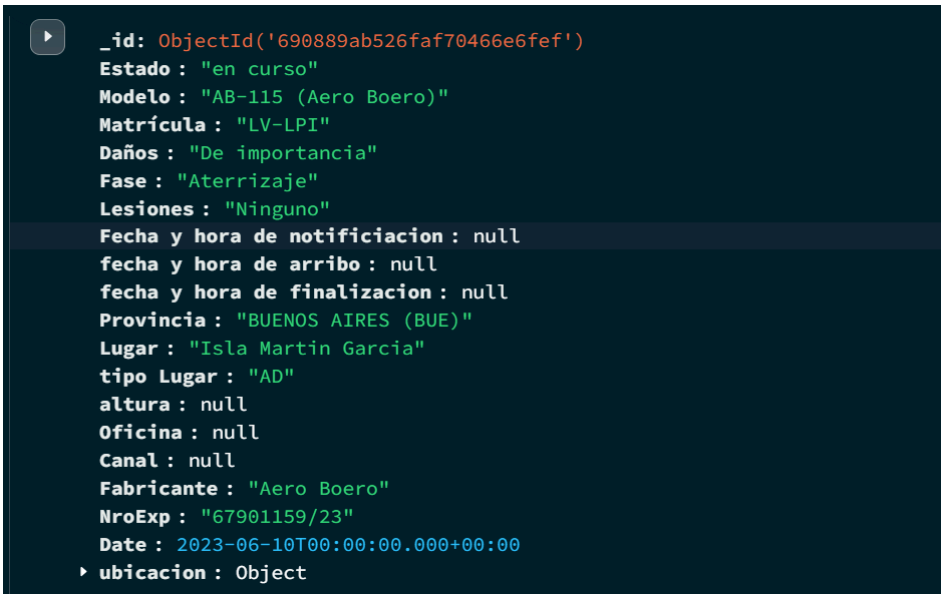

Imagen 2.23 Estructura de un documento

De esta manera se concluye la limpieza de la base de datos “Sucesos Aeronáuticos”, este proceso es super importante, ya que cuando se importaron los datos desde el archivo csv. ofrecido por Kaggle, muchos venían por defecto con un tipo de dato que no es funcional para el correcto funcionamiento del proyecto.

Las acciones antes descritas garantizan que la obtención de información a través de los datos se haga de una manera más eficiente y orientada a mejoras y optimizaciones futuras.

3. Problemática

Para el desarrollo de este informe, se crea una problemática la cual da lugar a las consultas que se desarrollan posteriormente.

En los últimos años, la aviación civil en Argentina ha experimentado un crecimiento sostenido en el tráfico aéreo, tanto en vuelos comerciales como en operaciones privadas y de instrucción. Sin embargo, este aumento ha venido acompañado de una persistencia en la ocurrencia de incidentes y accidentes aeronáuticos, muchos de los cuales podrían haberse evitado mediante una mejor comprensión de los factores contribuyentes.

A pesar de los esfuerzos regulatorios y de supervisión por parte de la Junta de Seguridad en el Transporte (JST) y otros organismos, no se cuenta con un sistema de análisis predictivo que permita identificar patrones recurrentes en los sucesos aeronáuticos. Esto limita la capacidad de anticipar riesgos y diseñar estrategias preventivas eficaces.

Líneas de consulta:

1. **Se necesita conocer cuáles son las provincias con mayor concentración de accidentes, ordenadas de mayor a menor.**

Comentarios

Para darle solución a esta consulta lo primero que se hace es agrupar los documentos por el campo de interés que es “Provincia”, luego se establece una variable “CantidadAccidentes” la cual a través del operador $sum, va a contar todos los accidentes que tiene cada Provincia. Posteriormente se con el operador $sort, se ordenan en sentido descendente (-1) y luego con $limit se limita los resultados arrojados a 3. Con esta lógica se garantiza obtener las tres provincias en las que más accidentes han ocurrido.

Código

(Pipeline de agregación con agrupación, ordenamiento y límite)

```
[{$group: {
  _id: "$Provincia",
  CantidadAccidentes: {
    $sum: 1
  }
}},
```

{$sort: {

CantidadAccidentes: -1

}},

{$limit: 3}]

Resultado

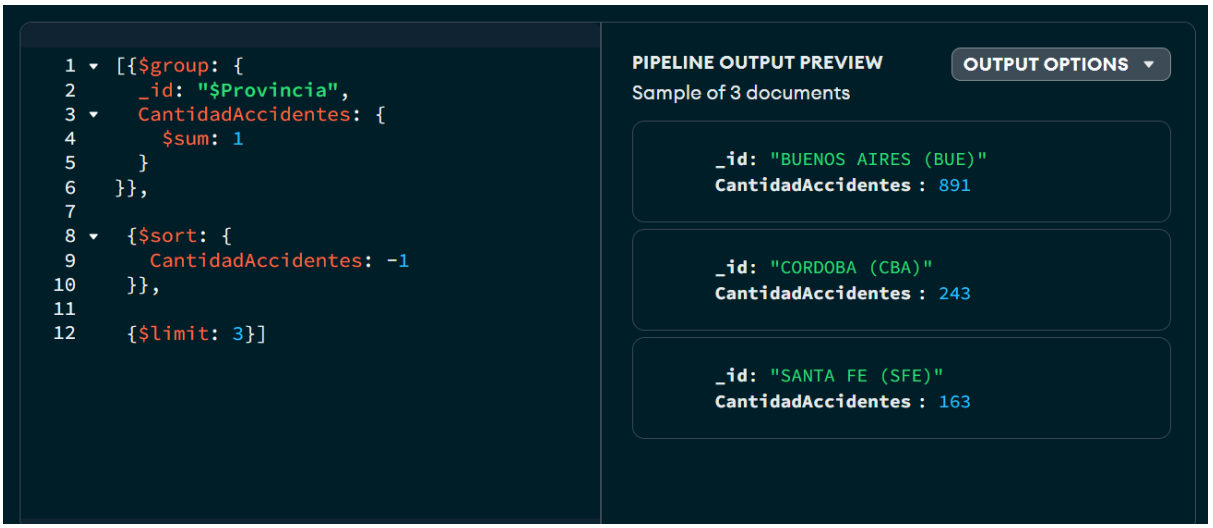

Imagen 3.1 Respuesta a consulta #1

2. **Se necesita conocer qué modelo de aeronaves está involucrado en más sucesos y cuál es su fabricante.**

Comentarios

Para darle respuesta a esta consulta lo primero que se hace es agrupar los documentos por modelo de avión y su fabricante. Luego de esto contamos cuentos sucesos tiene cada modelo de avión a través del operador $sum. Posteriormente, los resultados arrojados se ordenan y se limitan para así obtener el modelo de avión que más sucesos ha tenido.

Código

(Pipeline de agregación con agrupación compuesta, ordenamiento y selección del máximo)

[

{

$group: {

```
    _id: {modelo:"$Modelo",
          fabricante: "$Fabricante"} ,
    numSucesos: { $sum: 1 }
  }
},

{
  $sort: {
    numSucesos: -1
  }
},

{
  $limit: 1
},
```

Resultado

Imagen 3.2 Respuesta a consulta #2

3. **Se necesita conocer en qué periodo del año suceden más accidentes (invierno,verano,otoño,primavera).**

Comentarios

En un primer momento se tiene que crear una lógica capaz de determinar la estación del año partiendo de la fecha del suceso. Para esto se crea el campo “Estación” y luego se hace un switch en el cual, a través del operador $in, se determina en cuál estación entra cada mes. Luego, se agrupa por estación y se cuenta la cantidad de sucesos que han ocurrido en cada época del año.Por último se ordenan en sentido descendente para poder visualizar qué estación contiene mayor número de accidentes. El resultado final arroja que en el periodo de verano es donde más accidentes ocurren con un total de 632 sucesos.

Nota: La base de datos corresponde a Argentina, ubicada en el hemisferio sur, donde las estaciones ocurren en meses distintos a los de España. Esta diferencia estacional se considera explícitamente en el análisis.

Código

(Pipeline de agregación con clasificación temporal y conteo por categoría)

```
[

 {

  $addFields: {

   Estacion: {

    $switch:{

     branches: [

      {case:{$in:[{$month:"$Date"},[6,7,8]]}, then: "Invierno"},

      {case:{$in:[{$month:"$Date"},[9,10,11]]}, then: "Primavera"},

      {case:{$in:[{$month:"$Date"},[12,1,2]]}, then: "Verano" },

      {case:{$in:[{$month:"$Date"},[3,4,5]]}, then: "Otoño" }


     ],

     default: "Desconocido"
```

```
                                    }
                                }
                            }
                        },

                        {$group: {
                            _id: "$Estacion",
                            CantSucesos: {
                                $sum: 1
                            }
                        }},

                        {$sort: {
                            CantSucesos : -1
                        }}
                    ]
```

Resultado

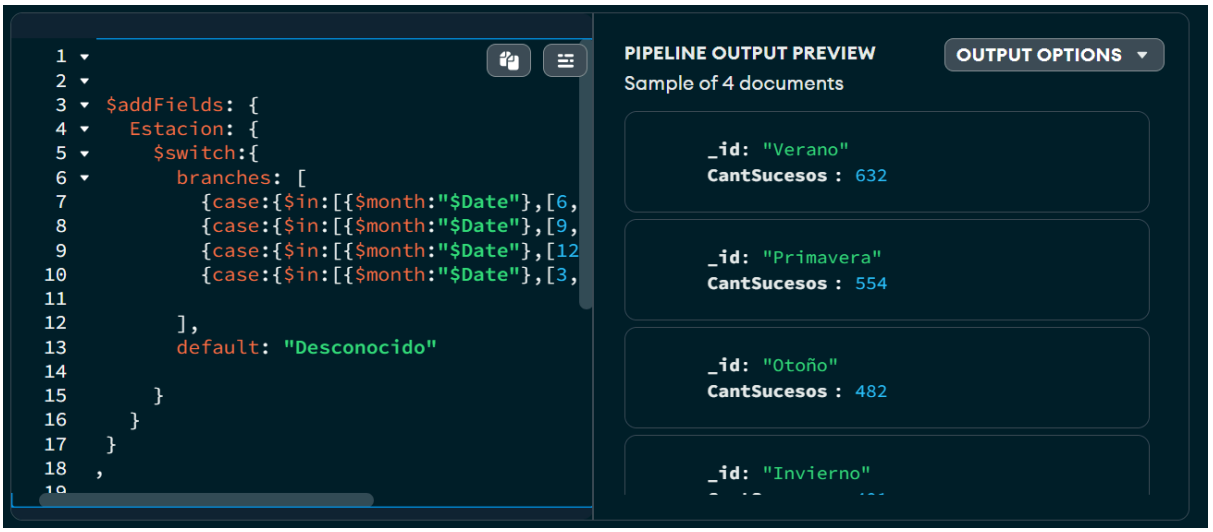

Imagen 3.3 Respuesta a consulta #3

4. **Se necesita conocer el % de incidentes con daños “De Importancia” por fabricante respecto al total de incidentes.**

Comentarios

En esta consulta, lo primero que se hace es utilizar $facet para ejecutar dos pipelines en paralelo. En un primer momento se obtiene el total de sucesos de la colección y luego, se filtran los documentos que contienen en el campo “Daños” la categoría “De importancia” y se agrupan por fabricante. Posteriormente se utiliza el $unwind para convertir los arrays generados en documentos individuales. En una última etapa, se utiliza $project para renombrar los campos y que se expresen en la salida de una manera más clara, se utilizan los operadores $multiply y $divide para poder sacar el % y luego, con $sort ordenamos los fabricantes por cantidad de incidentes “de importancia” de mayor a menor.

Código

(Pipeline de agregación con análisis paralelo, cálculo porcentual y ranking)

```
[
  {
    $facet: {
      total: [
        { $count: "totalGlobal" }
      ],
```


```
    importancia: [
      { $match: { Daños: "De importancia" } },
      {
        $group: {
          _id: "$Fabricante",
          totalImportancia: { $sum: 1 }
        }
      }
    ]
  }
},
{ $unwind: "$total" },
{ $unwind: "$importancia" },
{
  $project: {
    Fabricante: "$importancia._id",
    totalImportancia: "$importancia.totalImportancia",
    porcentaje: {
      $round: [
        {
          $multiply: [
            { $divide: ["$importancia.totalImportancia", "$total.totalGlobal"] },
            100
          ]
        },
```

```
      2
    ]
  }
}
},
{ $sort: { totalImportancia: -1 } }
]
```

Resultado

Imagen 3.4 Respuesta a consulta #4

5. **Se necesita conocer en qué año sucedieron más accidentes.**

Comentarios

Para obtener el año en que sucedieron más accidentes lo primero es agrupar los sucesos por año, a través del operador $year, se obtiene el año de la fecha. Una vez se agrupa así, se cuenta con $sum la cantidad de sucesos por año. Posteriormente, se ordenan con $sort y con $limit se garantiza obtener el año que más sucesos ocurrieron.

Código

(Pipeline de agregación con agrupación temporal, conteo y selección del máximo)

```
[{$group: {
 _id: {$year:"$Date"},
 Año: {
  $sum:1
 }
}},

{$sort: {
 Año: -1
}},

{$limit: 1}]
```

Resultado

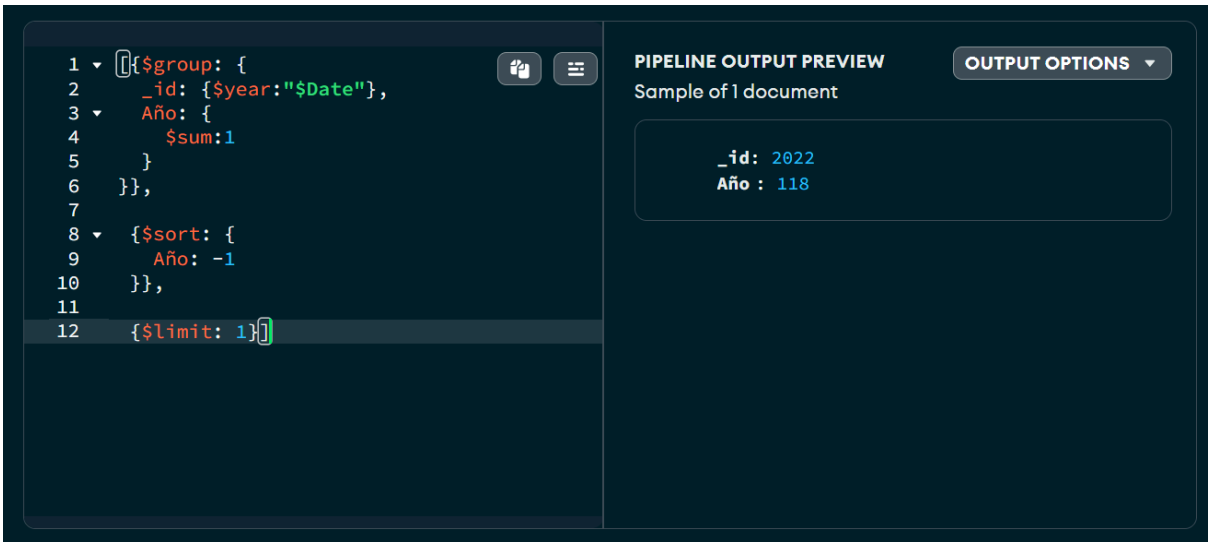

Imagen 3.5 Respuesta a consulta #5

6. **Se necesita conocer los sucesos sin lesiones pero con daños graves que ocurrieron en el invierno de 2019.**

Comentarios

Para realizar esta consulta lo primero es filtrar con $match los accidentes que cumplan con las características de ninguna lesión pero con daños de importancia. Luego, con $expr se garantiza poder utilizar expresiones de agregaciones dentro de una etapa. Seguido, se utiliza $and para sacar los sucesos que cumplan con ambas condiciones, la primera que sean del año 2019 y la segunda que sea del periodo de invierno (se utiliza los meses 6,7,8 porque esto es una base de datos de Argentina y en ese hemisferio estos meses corresponden al invierno). Para finalizar, con $count se cuenta la cantidad de documentos que cumplan con lo ante expuesto.

Código

(Pipeline de agregación con filtrado condicional y conteo)

```
[
  {
    $match: {
      Lesiones: "Ninguno",
      Daños: "De importancia",
      $expr: {
        $and: [
          { $eq: [{ $year: "$Date" }, 2019] },
          {
            $in: [{ $month: "$Date" }, [6, 7, 8]]
          }
        ]
      }
    }
  },
  {
    $count: "Total"
```

Resultado

Imagen 3.6 Respuesta a consulta #6

7. **Se necesita agrupar por ubicación redondeada para detectar zonas críticas.**

Comentarios

Con esta consulta se intenta buscar las zonas más críticas respecto a la cantidad de incidentes. Para lograr esto se empieza filtrando con $match todas las ubicaciones ya que existen documentos que no contienen coordenadas, es decir, su valor es nulo. Una vez que se garantiza que los documentos filtrados no tienen coordenadas nulas, se procede a redondear los valores de las coordenadas con $round para así crear grupos de coordenadas colindantes. Posteriormente, los documentos se agrupan según las coordenadas redondeadas, lo que permite identificar zonas con concentración de incidentes. Para cada grupo, se utiliza el operador $push para almacenar los identificadores de los sucesos en un array, facilitando la visualización de eventos colindantes en la salida. Por último, se ordenan en sentido descendente.

Código

(Pipeline de agregación con filtrado geoespacial, redondeo, agrupación y ranking)

```
    $match: {
      "ubicacion.coordinates": { $type: "array" },
      $expr: {
        $and: [
          { $ne: [ { $arrayElemAt: ["$ubicacion.coordinates", 0] }, null ] },
          { $ne: [ { $arrayElemAt: ["$ubicacion.coordinates", 1] }, null ] }
        ]
      }
    }
  },
  {
    $addFields: {
      lonRedondeada: {
        $round: [ { $arrayElemAt: ["$ubicacion.coordinates", 0] }, 2 ]
      },
      latRedondeada: {
        $round: [ { $arrayElemAt: ["$ubicacion.coordinates", 1] }, 2 ]
      }
    }
  },
  {
    $group: {
      _id: {
        lon: "$lonRedondeada",
        lat: "$latRedondeada"
```

```
    },
    totalIncidentes: { $sum: 1 },
    ejemplos: { $push: "$_id" }
  }
},
{
  $sort: { totalIncidentes: -1 }
}
]
```

Resultado

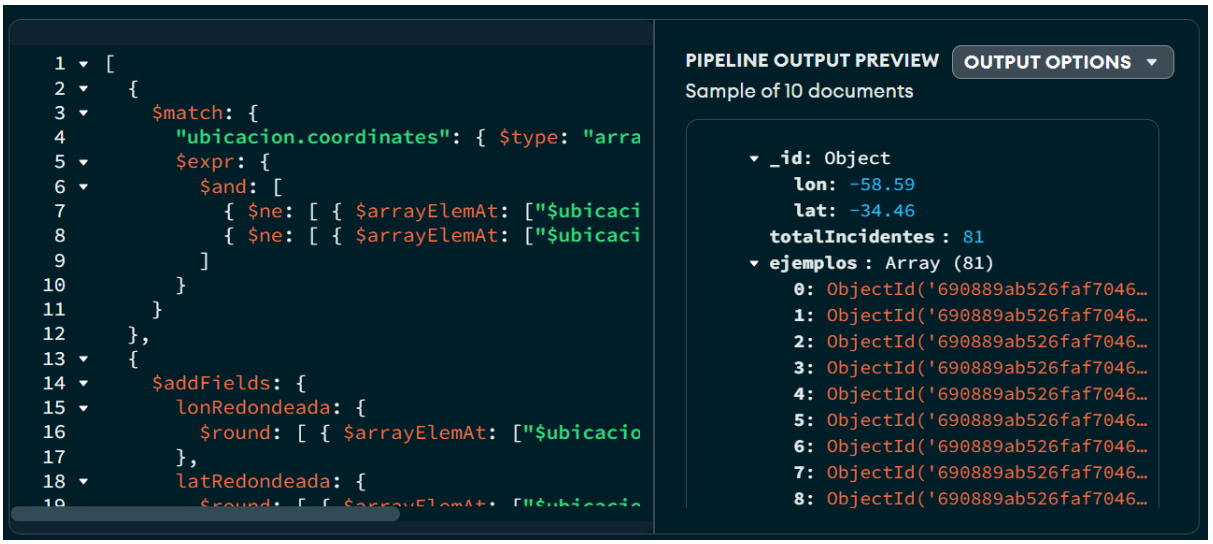

Imagen 3.7 Respuesta a consulta #7

8. **Se necesita que si un caso, lleva más de 1 año “en curso”, se cierre y por tanto se modifique la fecha de finalización de expediente**

Comentarios

La consulta tiene como objetivo actualizar automáticamente el estado de los sucesos que llevan más de un año activos. Para ello, se define una condición que evalúa si el campo “Estado” es “en curso”y si la diferencia entre la fecha del suceso (“Date”) y la fecha actual supera un año. Si ambas condiciones se cumplen, se utiliza $set para modificar el valor de “Estado” a “Cerrado”.

Posteriormente, se actualiza el campo “fecha y hora de finalización” asignándole la fecha actual ($$NOW), reflejando así el cierre efectivo del caso. Finalmente, se aplica $merge para persistir los cambios directamente en la colección original.

Código

(Pipeline de agregación con lógica condicional temporal y persistencia)

```
[{
  "$set": {
    "Estado": {
      "$cond": {
        "if": {
          "$and": [
            { "$eq": [ { "$getField": "Estado" }, "en curso" ] },
            {
              "$gt": [
                {
                  "$dateDiff": {
                    "startDate": { "$getField": "Date" },
                    "endDate": "$$NOW",
                    "unit": "year"
                  }
                },
                1
              ]
            }
          ]
        },
```

```
        "then": "Cerrado",
        "else": { "$getField": "Estado" }
      }
    },
    "fecha y hora de finalización": {
      "$cond": {
        "if": {
          "$and": [
            { "$eq": [ { "$getField": "Estado" }, "en curso" ] },
            {
              "$gt": [
                {
                  "$dateDiff": {
                    "startDate": { "$getField": "Date" },
                    "endDate": "$$NOW",
                    "unit": "year"
                  }
                },
                1
              ]
            }
          ]
        },
        "then": "$$NOW",
        "else": { "$getField": "fecha y hora de finalización" }
```

```
    }
  }
},
},
{$merge: {
  into: 'Argentina',
  whenMatched: 'merge',
  whenNotMatched: 'discard'
}}]
```

Resultado

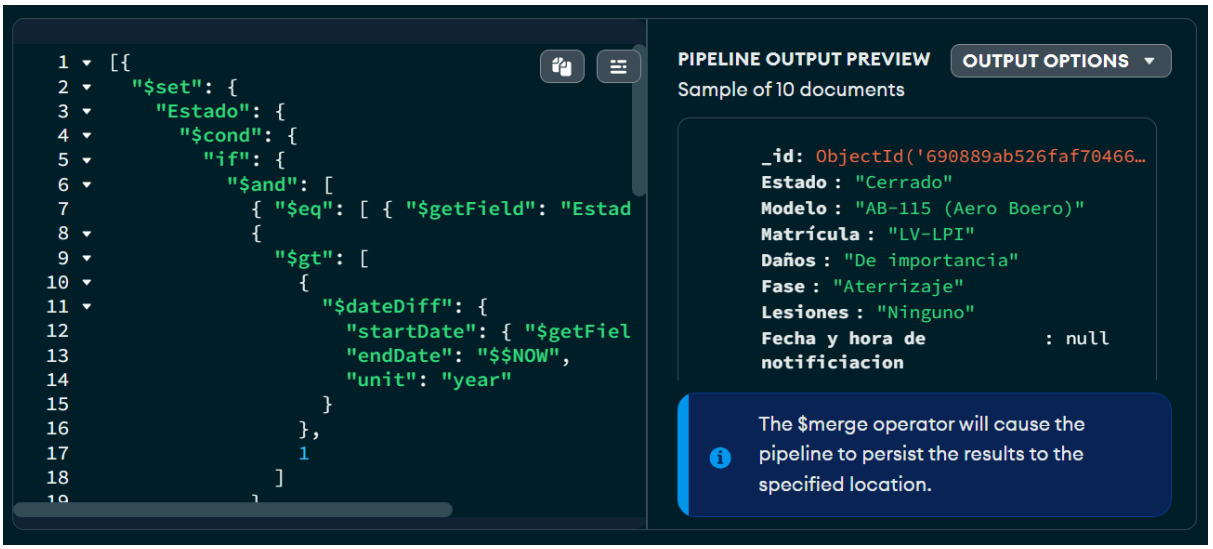

Imagen 3.8 Respuesta a consulta #8

9. **Se necesita la provincia con menor tiempo promedio de reacción entre notificación y arribo. Para esto es imprescindible rellenar los valores de "fecha de notificación" y "fecha de arribo".**

<u>1ra Parte</u>

Comentarios

Este código sirve para generar dos fechas aleatorias a partir de una fecha base llamada "fecha". Primero, calcula la "Fecha y hora de notificación" sumando entre 0 y 360 minutos (es decir, hasta 6 horas) a la fecha original. Para lograrlo, usa el operador $rand, que genera un número aleatorio entre 0 y 1, lo multiplica por 360 con $multiply, y luego lo redondea hacia abajo con $floor para obtener un número entero de minutos. Luego, para la "Fecha y hora de arribo", vuelve a partir de la misma fecha, le suma otra cantidad aleatoria de minutos como si fuera la notificación, y sobre esa nueva fecha le agrega entre 5 y 180 minutos más. Esto garantiza que el arribo siempre ocurra después de la notificación, simulando un tiempo de respuesta realista.

Código

(Pipeline de agregación - Actualización,)

```
[{
  "$set": {
    "Fecha y hora de notificiacion": {
      "$dateAdd": {
        "startDate": "$Date",
        "unit": "minute",
        "amount": {
          "$floor": {
            "$multiply": [ { "$rand": {} }, 360 ]  // 0–6 horas en minutos
          }
        }
      }
    },
    "fecha y hora de arribo": {
      "$dateAdd": {
        "startDate": {
          "$dateAdd": {
            "startDate": "$Date",
```

```
          "unit": "minute",
          "amount": {
            "$floor": {
              "$multiply": [ { "$rand": {} }, 360 ]
            }
          }
        }
      },
      "unit": "minute",
      "amount": {
        "$add": [
          5,
          {
            "$floor": {
              "$multiply": [ { "$rand": {} }, 175 ]  // 5–180 minutos
            }
          }
        ]
      }
    }
  }
},
{$merge: {
  into: 'Argentina',
```

```
  whenMatched: 'merge',
  whenNotMatched: 'discard'
}}]
```

Resultado

Imagen 3.9 Respuesta a consulta #9 Parte 1

Comentarios

Para esta parte del código, lo que se busca es sacar la diferencia de minutos entre la fecha de notificación y la fecha de arribo. Para lograr esto se utiliza el operador $dateDiff, una vez obtenida la diferencia de minutos, se procede a agrupar por provincias y mostrar el promedio de minutos, el cual se saca atraves de $avg.

Código

(Pipeline de agregación con cálculo temporal, agrupación y ordenamiento)

```
[
  {
    "$project": {
      "Provincia": 1,
      "minutos_diferencia": {
        "$dateDiff": {
```

```
        "startDate": "$Fecha y hora de notificiacion",
        "endDate": "$fecha y hora de arribo",
        "unit": "minute"
    }
  }
 }
},
{
  "$group": {
    "_id": "$Provincia",
    "promedio_minutos": { "$avg": "$minutos_diferencia" },
  }
},
{
  "$sort": { "promedio_minutos": -1 }
}
]
```

Resultado

Imagen 3.10 Respuesta a consulta #9 Parte 2

10. **Antes de registrar un nuevo accidente en Mendoza el 5 de enero de 2024, verificar que no exista otro con el mismo modelo, fecha y ubicación. Si no existe, insertar el nuevo documento.**

Comentarios

Para ejecutar esta consulta, primero se verifica si ya existe un documento con la misma información mediante find(). Si no se encuentra coincidencia, se procede a insertar el nuevo registro utilizando insertOne(), estableciendo todos los campos relevantes del accidente, incluyendo fechas, ubicación geoespacial, estado, fabricante y detalles del evento.

Código

(Consultar la existencia de un documento con esas características)

```
db.Argentina.find({ Modelo: "Hilux", Fecha: "05/01/2024", "ubicacion.type": "Point","ubicacion.coordinates": [-68.8272, 32.8908]})
```

(Insertar el nuevo accidente)

```
db.Argentina.insertOne({
 Estado: "Cerrado",
 Modelo: "Hilux",
 Matrícula: "AA-123-BB",
 Daños: "De importancia",
 Fase: "Circulación urbana",
 Lesiones: "Leves",
 "Fecha y hora de notificiacion": ISODate("2024-01-05T08:30:00.000Z"),
 "fecha y hora de arribo": ISODate("2024-01-05T09:00:00.000Z"),
```

```
  Provincia: "MENDOZA (MZA)",
  Lugar: "Av. San Martín y Belgrano",
  "tipo Lugar": "Vía pública",
  altura: null,
  Oficina: null,
  Canal: null,
  Fabricante: "Toyota",
  NroExp: "78945612/24",
  Date: ISODate("2024-01-05T00:00:00.000Z"),
  ubicacion: {
    type: "Point",
    coordinates: [-68.8272, 32.8908]
  },
  "fecha y hora de finalización": ISODate("2024-01-05T10:15:00.000Z")
})
```

Resultado

(Consultar la existencia de un documento con esas características)

```
> db.Argentina.find({Modelo: "Hilux",Fecha: "05/01/2024","ubicacion.type": "Point","ubicacion.coordi
<
SucesosAeronautica >
```

Imagen 3.11 Respuesta a consulta #10 Parte 1

```
>_MONGOSH
    "Fecha y hora de notificiacion": ISODate("2024-01-05T08:30:00.000Z"),
    "fecha y hora de arribo": ISODate("2024-01-05T09:00:00.000Z"),
    Provincia: "MENDOZA (MZA)",
    Lugar: "Av. San Martín y Belgrano",
    "tipo Lugar": "Vía pública",
    altura: null,
    Oficina: null,
    Canal: null,
    Fabricante: "Toyota",
    NroExp: "78945612/24",
    Date: ISODate("2024-01-05T00:00:00.000Z"),
    ubicacion: {
      type: "Point",
      coordinates: [-68.8272, 32.8908]
    },
    "fecha y hora de finalización": ISODate("2024-01-05T10:15:00.000Z")
  })
< {
    acknowledged: true,
    insertedId: ObjectId('691b19761917c2dc51083461')
  }
SucesosAeronautica >
```

Imagen 3.12 Respuesta a consulta #10 Parte 2

4.Conclusiones

A lo largo del desarrollo de esta práctica se abordó de forma integral el ciclo completo de trabajo con una base de datos NoSQL en MongoDB, partiendo desde la importación de datos crudos hasta la ejecución de consultas analíticas. El proceso comenzó con la limpieza estructural del dataset, corrigiendo inconsistencias como valores no estándar ("S/D") en campos clave como altura, oficina, canal y fechas. Estos fueron transformados a null mediante actualizaciones masivas, lo que permitió aplicar operadores de tipo y fecha sin errores y garantizó la compatibilidad con futuras transformaciones.

Posteriormente, se normalizó el campo "Lat - Lon" mediante un pipeline de agregación que lo convirtió en un objeto GeoJSON, habilitando así análisis geoespaciales conforme al estándar de MongoDB. También se transformó el campo "Fecha" de string a tipo Date, y se creó un nuevo campo "Estación" utilizando $switch y $month, lo que permitió clasificar los eventos por estación del año considerando la ubicación geográfica de Argentina en el hemisferio sur. Estas transformaciones enriquecieron la semántica temporal de los documentos y habilitaron cálculos cronológicos como el tiempo de respuesta entre notificación y arribo.

En cuanto a las consultas analíticas, se identificaron las provincias con mayor número de accidentes, destacando Buenos Aires, Córdoba y Santa Fe. También se determinó que el modelo de aeronave con más sucesos fue el PA-11 de Piper, y se concluyó que el verano es la estación con mayor concentración de accidentes. Estas consultas se resolvieron mediante operadores como $group, $sum, $sort, $limit y $facet, demostrando cómo MongoDB puede ser utilizado como herramienta de análisis exploratorio y estadístico.

Además, se realizaron inserciones controladas de nuevos documentos, aplicando verificaciones previas con find() para evitar duplicados. Los registros fueron estructurados con campos completos como fechas en formato ISO, ubicación geoespacial, y metadatos operacionales. Estas inserciones se integraron con lógica de transformación y persistencia condicional mediante $merge, simulando flujos reales de actualización de datos y consolidación de eventos.

En conjunto, los análisis realizados demuestran un dominio técnico de MongoDB, aplicando operadores de transformación, limpieza, proyección, filtrado y agregación con precisión. La práctica no solo cumplió con los requisitos académicos, sino que también evidenció la capacidad de aplicar MongoDB en escenarios reales de gestión de sucesos aeronáuticos, con un enfoque orientado a la calidad de datos y eficiencia operativa.