Universidad Carlos III

Arquitectura de Datos

Curso 2024-25

Práctica 1.2

Migración de Base de Datos a MongoDB

Ingeniería Informática, Cuarto curso

Adrián Fernández Galán (NIA: 100472182, e-mail: 100472182@alumnos.uc3m.es)
César López Mantecón (NIA: 100472092, e-mail: 100472092@alumnos.uc3m.es)
Manuel Gómez-Plana Rodríguez (NIA: 100472310, e-mail: 100472310@alumnos.uc3m.es)

Prof. Lourdes Moreno López
Grupo: 81

Índice

1. Introducción

En este documento se recoge el desarrollo de la segunda práctica de la asignatura Arquitectura de datos. A continuación, se especifica el preprocesado de los datos usando la biblioteca *Pandas* de Python, la realización de la validación de esquemas de mongoDB y la carga de los csvs limpios a esta herramienta.

Para el desarrollo de la práctica, se migrará una base de datos para la gestión de áreas recreativas a un modelo no relacional a través de `MongoDB`. De esta forma se podrán aprovechar las ventajas de este tipo de bases de datos en flexibilidad, escalabilidad horinzontal y velocidad.

Durante el proceso de migración se seguirá la siguiente metodología:

1. **Limpieza de los datos**: la base de datos previa a la migración está contenida en ficheros `csv`. Esto nos permite usar la herramienta *Pandas* y el lenguaje de programación `Python` para preprocesar y limpiar los datos.

2. **Importación y trasnformación en** `MongoDB`: Se importarán los datos a `Mongo` y se harán las transformaciones necesarias para que los datos tengan el tipo correcto.

3. **Validación de esquemas**: Se aplicará un esquema de validación a cada uno de los documentos para asegurar que los datos tienen el formato correcto.

4. **Creación de agregados**: Se crearán los agregados especificados en el enunciado, calculando todos los atributos derivados necesarios.

Con todo lo anterior, la migración será completa y los datos estarán listos para su uso. Posteriormente, se desarrollará una estructura de *cluster* para una gestión más eficiente de los datos y las consultas.

A continuación se expone cómo ejecutar el preprocesado correctamente.

1.1. Cómo ejecutar el preprocesado

Para facilitar la ejecución del preprocesado se ha desarrollado un *script* que limpia los datos y realiza la migración completa a `MongoDB`. Este *script* está contenido en el archivo `import_data.sh` y se debe llamar desde la carpeta donde está contenido:

```
# Desde la carpeta entregable2
bash import_data.sh
```

Listing 1: Comando para realizar el preprocesado

Es importante descatar que para el correcto funcionamiento de este script deben existir dos carpetas:

- `csvs`: esta carpeta debe contener los ficheros con los datos en sucio.
- `output`: en esta carpeta se almacenarán los datos en limpio para su posterior inserción en `Mongo`.

2. Preprocesado de Datos

En este apartado se describen las técnicas para el preprocesado de los datos con el fin de prepararlos para su inserción en `MongoDB`. Para este preprocesado se ha usado la librería de Python *Pandas*, siguiendo una metodología común para cada fichero que constaba de los siguientes pasos:

1. Se carga el csv a limpiar en un *dataframe* de *Pandas*
2. Se aplican técnicas de preprocesado como la normalización de los strings o imputación genérica de *missing values*
3. Se escribe el csv limpio a partir del *dataframe* preprocesado

2.1. Técnicas de preprocesado comunes

Se han aplicado técnicas de preprocesado que son comunes a todos los ficheros, independientemente de cual se quiera limpiar. A continuación, se listan y explican las técnias más usadas:

- **Normalización de strings**:

  Todos los datos categóricos han sido normalizados de manera que siempre estén en mayúscula y sin tildes. Esto se ha realizado empleando la siguiente función:

```
def change_accents(word):
    if type(word) != str:
        return
    for letter in range(len(word)):
        if word[letter] == "Á":
            word = word[0:letter] + "A" + word[letter + 1:]
        if word[letter] == "É":
            word = word[0:letter] + "E" + word[letter + 1:]
        if word[letter] == "Í":
            word = word[0:letter] + "I" + word[letter + 1:]
        if word[letter] == "Ó":
            word = word[0:letter] + "O" + word[letter + 1:]
        if word[letter] == "Ú":
            word = word[0:letter] + "U" + word[letter + 1:]
    return word
```

Listing 2: Eliminación de tildes

  La cuál se llama de la siguiente manera:

```
dataframe.loc[indice, "COLUMNA_A_CAMBIAR"] = change_accents(
   valor["COLUMNA_A_CAMBIAR"].upper())
```

Listing 3: Capitalización de strings y eliminacion de tildes

- **Rellenado de fechas**

  Las fechas que tenían el valor "fecha desconocida" o que no tenían valor han sido rellenadas con el valor "01/01/1970" a través del siguiente código:

```
for indice, value in dataframe.iterrows():
    if not pd.notna(value["FECHA_INSTALACION"]) or value["
       FECHA_INSTALACION"] == "fecha_incorrecta":
        dataframe.loc[indice, "FECHA_INSTALACION"] = "
           1970-01-01T00:00:00Z"
```

Listing 4: Imputacion de fechas incorrectas o faltantes

  Se ha seleccionado este valor por tratarse de la *fecha UNIX*. De esta forma es un valor fácilmente reconocible que no inclumple la validación de esquemas que implementaremos en `MongoDB`.

- **Formateo de fechas**

  Todas las fechas han sido formateadas al formato de mongo a través del siguiente código:

```
dataframe["COLUMNA_A_CAMBIAR"] = pd.to_datetime(dataframe["
    COLUMNA_A_CAMBIAR"], format="mixed", dayfirst=True, utc=True
    ).dt.strftime('%Y-%m-%dT%H:%M:%SZ')
```

Listing 5: Formateo de fechas"

  Se ha seleccionado este valor por tratarse de la *fecha UNIX*. De esta forma es un valor fácilmente reconocible que no inclumple la validación

- **Rellenado genérico de *missing values*.**

  Las columnas que tenían valores nulos que no ha sido posible imputar han sido rellenadas con un valor fácilmente indetificable que sigue el formato "NOM-COL_DESCONOCIDO_ID-FILA" mediante la siguiente función:

```
def fill_missing_tipo(row,column,string_missing):
    if pd.isnull(row[column]):
        return f'{string_missing}_{row["ID"]}'
    return row[column]
```

Listing 6: Inputacion de valores faltantes desconocidos

  Esta función se llama de la siguiente manera:

```
df["COLUMNA"] = df.apply(lambda row: fill_missing_tipo(row, "
    COLUMA", "COLUMNA_DESCONOCIDO"), axis=1)
```

Listing 7: llamada a la función para rellenar valores faltantes

2.2. Preprocesado de los csvs en Python

En este apartado se explican las técnicas y procedimientos para el preprocesado de cada uno de los csvs sucios desarrollado en Pyhon usando la libería *pandas*.

2.2.1. Juegos

Para el fichero *JuegosSucio.csv* se han aplicado las siguientes transformaciones:

1. Apertura y carga de valores en dataframe
2. Imputación de valores en *DISTRITO* y *CODIGO_DISTRITO*: se han relacionado valores de distrito con códigos de forma unívoca, rellenando los valores faltantes en caso necesario. Esto se hace mediante la llamada a la función `imput_missing_district`.
3. Imputación de valores en *TIPO_VIA*, *NOMBRE_VIA*, y *NUM_VIA*: se ha extraído toda la información posible de *DIRECCION_AUX* y rellenado los campos anteriores en caso posible.
4. Fusión con *areas_limpias.csv*: se han asociado los juegos a un área mediante los campos *CODIGO_INTERNO* y *NDP*. Esto se ha traducido en una columna nueva en la tabla de *juegos* con el ID del área al que pertenece. Adicionalmente se ha completado la información faltante de cada tabla con información del otro en los casos posibles.

2.2.2. Áreas

Para el fichero *AreasSucio.csv* se ha aplicado el siguiente proceso:

1. Apertura y carga de valores en dataframe
2. Normalización de la columna *DESC_CLASIFICACION*: mediante el paso a mayúsculas y la eliminación de los acentos de toda la columna
3. Normalización de la columna *BARRIO*: mediante el paso a mayúsculas y eliminación de los acentos de toda la columna, así como la obtención de los valores de *COD_POSTAL* por barrio para su posterior imputación siempre y cuando sus valores no sean nulos
4. Imputación de los valores faltantes de *COD_POSTAL*: mediante la identificación del barrio y la imputación del código postal identificado por el barrio
5. Normalización de la columna *DISTRITO*: mediante el paso a mayúsculas y eliminación de los acentos de toda la columna, así como la obtención de los valores de *COD_DISTRITO* por distrito y viceversa para su posterior imputación, siempre y cuando sus valores no sean nulos
6. Imputación de los valores faltantes de *COD_DISTRITO* y *DISTRITO*: mediante la identificación del código buscando por distrito cuando este no es nulo y viceversa
7. Imputación de valores en *TIPO_VIA*, *NOMBRE_VIA*, y *NUM_VIA*: se ha extraído toda la información posible de *DIRECCION_AUX* y rellenado los campos anteriores en caso posible siguiendo los siguientes pasos:
   - Si no hay *TIPO_VIA*, se extrae el tipo de la dirección auxiliar. Si esta no existe, se escribe el valor "tipo_desconocido_ID"
   - Si no hay *NOM_VIA*, se extrae el nombre de la dirección auxiliar. Si esta no existe, se escribe el valor "NOMBRE_DESCONOCIDO_ID"
   - Si no hay *NUM_VIA*, se extrae el número de la dirección auxiliar. Aquí, puede haber tres opciones:
     - Existe solo un número en la dirección auxiliar, que tiene la forma "VALDEBERNARDO, 000037". En este caso, simplemente se extrae y se castea a entero para eliminar los 0's innecesarios
     - Existen más de un número en la dirección auxiliar, que tiene la forma "V · VIA LÍMITE 115 , ASCENDIENTE 1: 064 · ALMENAR". En este caso, se extrae el primer número y se interpreta como el número de la vía

        - Si no existen números en la dirección auxiliar, que tiene la forma "PARQUE ROMA". Aquí, simplemente se escribe el valor "NUMERO_DESCONOCIDO_ID"
    - Tras sacar los datos de la dirección auxiliar, se borra esta columna de la instancia imputada
8. Formateo de *FECHA_INSTALACION*: mediante el paso de la fecha al formato de mongo
9. Imputación de valores faltantes en *CODIGO_INTERNO*: mediante el rellenado de estos datos con los valores "CÓDIGO_INTERNO_ID_DESCONOCIDO".
10. Normalización de la columna *tipo*: mediante el paso a mayúsculas y eliminación de los acentos de toda la columna
11. Escritura del csv limpio final

2.2.3. Encuestas satisfacción

Para el fichero *EncuestasSatisfaccion.csv* se ha aplicado el siguiente proceso:

1. Apertura y carga de valores en dataframe
2. Formateo de *FECHA*: mediante el paso de la fecha al formato de mongo
3. Normalización de la columna *COMENTARIOS*: mediante el paso a mayúsculas de todos los valores
4. Renombrado de la columna *ID* a "_id" para su posterior inserción en mongo
5. Escritura del csv limpio final

2.2.4. Incidencias Usuario

Para el fichero *IncidenciasUsuariosSucio.csv* se ha aplicado el siguiente proceso:

1. Apertura y carga de valores en dataframe
2. Normalización de la columna *TIPO_INCIDENCIA*: mediante el paso a mayúsculas de todos los valores
3. Formateo de *FECHA_REPORTE*: mediante el paso de la fecha al formato de mongo
4. Normalización de la columna *ESTADO*: mediante el paso a mayúsculas de todos los valores
5. Renombrado de la columna *ID* a "_id" para su posterior inserción en mongo
6. Eliminación de los usuarios replicados: debido a que existen usuarios con la misma información, salvo el correo, se ha decidido eliminar los usuarios duplicados
7. Escritura del csv limpio final

2.2.5. Incidencias Seguridad

Para el fichero *IncidentesSeguridadSucio.csv* se ha aplicado el siguiente proceso:

1. Apertura y carga de valores en dataframe
2. Formateo de *FECHA_REPORTE*: mediante el paso de la fecha al formato de mongo
3. Normalización de la columna *TIPO_INCIDENTE*: mediante el paso a mayúsculas y la eliminación de los acentos de toda la columna
4. Normalización de la columna *GRAVEDAD*: mediante el paso a mayúsculas y la eliminación de los acentos de toda la columna
5. Renombrado de la columna *ID* a "_id" para su posterior inserción en mongo
6. Escritura del csv limpio final

2.2.6. Mantenimiento

Para el fichero *MantenimientoSucio.csv* se ha aplicado el siguiente proceso:

1. Apertura y carga de valores en dataframe
2. Normalización de la columna *TIPO_INTERVENCION*: mediante el paso a mayúsculas de todos los valores
3. Formateo de *FECHA_INTERVENCION*: mediante el paso de la fecha al formato de mongo
4. Normalización de la columna *ESTADO_PREVIO*: mediante el paso a mayúsculas de todos los valores
5. Normalización de la columna *ESTADO_POSTERIOR*: mediante el paso a mayúsculas de todos los valores
6. Normalización de la columna *Tipo*: mediante el rellenado de los missing values con el valor "TIPO_ID_DESCONOCIDO" y su paso a mayúsculas de todos los valores
7. Normalización de la columna *Comentarios*: mediante el rellenado de los missing values con el valor "COMENTARIO_ID_DESCONOCIDO" y su paso a mayúsculas de todos los valores
8. Cambio de la columna *ID* para su posterior uso en los agregado en mongo. Esto se hace mediante la extracción el número del id y su concatenación a la cadena "MNT-", rellenando con ceros hasta que el número tenga cinco dígitos. Esto se realiza de la siguiente manera:

```
for indice, value in df.iterrows():
    id_antiguo = value["ID"]
    indice_numero = id_antiguo.index(",")
    indice_nuevo = id_antiguo[1: indice_numero]
    while len(indice_nuevo) < 5:
        indice_nuevo = "0" + indice_nuevo
    df.loc[indice, "ID"] = "MNT-" + indice_nuevo
```

9. Renombrado de la columna *ID* a "_id" para su posterior inserción en mongo
10. Eliminación de los mantenimientos replicados: debido a que existen mantenimientos con el mismo "_id", se ha decidido eliminar los valores duplicados
11. Escritura del csv limpio final

2.2.7. Meteo24

El fichero *meteo24.csv* se ha restructurado para conseguir el formato pedido, el proceso de reestructuración es el siguiente:

1. Se ha creado un nuevo dataframe con las siguientes columnas: *Id*, *Fecha*, *Temperatura*, *Precipitación*, *Viento* y *Estación*.
2. Se ha creado un diccionario en python que permita asociar el código de la magnitud con la propia magnitud con tan solo las magnitudes que nos interesan.
3. Para cada fila de meteo24 se ha tomado el valor de la magnitud, comprobando que era una de las magnitudes que nos interesan, el código de la estación, el valor del año y del mes, y se ha iterado por cada columna que corresponde a un día del mes
4. Para cada valor de día se creaba una fecha completa (%DD, %MM, %AAAA) y se tomaba el valor de la columna de ese día para esa magnitud. Con estos datos se rellena el dataframe nuevo de la siguiente manera.
    - Si esa fecha y estación no existen en el nuevo dataframe se crea una nueva fila y se rellena con únicamente la magnitud que se ha encontrado en ese momento, por lo que se crea una fila con magnitudes sin valor a las que se tendrá que esperar para dar valor.

- Si esa fecha y estación ya existe en el nuevo dataframe se rellena la fila existente con el nuevo valor de la magnitud.

5. Tras pasar por todas las filas de *meteo24.csv* se transforma la columna *Viento* a un booleano, dado que la magnitud que se nos pide es si el viento es fuerte o no, sin embargo a través de meteo conocemos la velocidad de este.

2.2.8. Usuarios

El fichero *UsuariosSucio.csv* se ha restructurado para conseguir el formato pedido, el proceso de reestructuración es el siguiente:

1. Apertura y carga de valores en dataframe
2. Normalización de la columna *NOMBRE*: mediante el paso a mayúsculas de todos los valores
3. Normalización de la columna *EMAIL*: mediante el paso a mayúsculas de todos los valores y el relleno de los missing values con el valor "EMAIL_ID_DESCONOCIDO"
4. Normalización de la columna *TELEFONO* mediante la eliminación del prefijo y de los espacios en blanco. Esto se realiza con la siguiente función:

```
def format_phone_number(phone):
    phone = phone.replace(" ", "")
    if phone.startswith("+34"):
        phone = phone[3:]
    if phone.startswith("34"):
        phone = phone[2:]
    return phone
```

5. Renombrado de la columna *NIF* a "_id" para su posterior inserción en mongo
6. Escritura del csv limpio final

2.3. Importación de datos a MongoDB

Los datos se han trasladado a mongo tras el primer preprocesado en `Python` a través del siguiente *script*:

```
mongoimport --db entregable2 --collection areas --type csv --file
    ../output/areas_limpias.csv --headerline
mongoimport --db entregable2 --collection juegos --type csv --file
    ../output/juegos_limpio.csv --headerline
mongoimport --db entregable2 --collection encuestas_satisfaccion --
    type csv --file ../output/encuestas_satisfaccion_limpio.csv --
    headerline
mongoimport --db entregable2 --collection
    estaciones_meteo_codigo_postal --type csv --file ../output/
    estaciones_meteo_codigo_postal.csv --headerline
mongoimport --db entregable2 --collection incidencias_usuarios --
    type csv --file ../output/incidencias_usuarios_limpio.csv --
    headerline
mongoimport --db entregable2 --collection incidentes_seguridad --
    type csv --file ../output/incidentes_seguridad_limpio.csv --
    headerline
mongoimport --db entregable2 --collection mantenimiento --type csv
    --file ../output/mantenimiento_limpio.csv --headerline
mongoimport --db entregable2 --collection meteo24 --type csv --file
     ../output/meteo24_limpio.csv --headerline
mongoimport --db entregable2 --collection usuarios --type csv --
    file ../output/usuarios_limpios.csv --headerline
```

Listing 8: Script para la importación en Mongo

De esta forma se insertan todos los registros en `Mongo`, cada uno en una colección distinta.

2.4. Preprocesado en MongoDB

Una vez que el grueso de la limpieza se ha realizado en Python, todavía existen algunos atributos con problemas que hay que resolver antes de se inserción en MongoDB. Es por ello que en este apartado se explica el preprocesado realizado en MongoDB.

2.4.1. Cambio de tipo

Algunos de los atributos de los csvs tienen un tipado incorrecto cuando se insertan en MongoDB. Es por ello que, tras su inserción, se le cambia el tipo a todos los atributos que lo necesiteb. Esto se realiza con el operador *addFields*, añadiendo una nueva columna con el mismo nombre que la que queremos cambiar de tipo. Luego, se emplea el operador *convert* para cambiar al tipo que se desee. Este proceso se ve en el siguiente código:

```
$addFields: {
    COD_DISTRITO: {
        $convert: {
            input: "$COD_DISTRITO",
            to: "int",
            onError: null,
            onNull: null
        }
    },
}
```

Listing 9: Cambio de tipo genérico

Todos los atributos que han necesitado un cambio son:

CSV	Atributo	Tipo nuevo
Áreas	COD_POSTAL	int
Áreas	COD_DISTRITO	int
Áreas	LATITUD	string
Áreas	LONGITUD	string
Áreas	NUM_VIA	string
Áreas	NUM_VIA	string
Juegos	COD_DISTRITO	int
Juegos	COD_POSTAL	string
Juegos	LATITUD	string
Juegos	LONGITUD	string
Juegos	MODELO	string
Juegos	ACCESIBLE	bool
Juegos	NUM_VIA	string
Encuestas Satisfaccion	AreaRecreativaID	string
Usuarios	_id	string
meteo24	_id	string
meteo24	VIENTO	bool

Cuadro 1: Atributos y sus nuevos tipos

2.4.2. Obtención de fechas a través de strings

De la misma manera, todas las fechas de todos los csvs son interpretadas erróneamenta al introducir los datos en mongo. Todas las fechas se interpretan como strings, así que deben ser cambiadas al tipo *Date*. Esto se realiza con el operador *addFields* que, de manera similar al apartado anterior, sustituye el atributo antiguo por el homónimo nuevo con su tipo corregido. También se usa el operador *dateFromString*, que devuelve una fecha ajustada al formato especificado. Todo esto se realiza mediante el siguiente código:

```
$addFields: {
    FECHA_REPORTE: {
        $dateFromString: {
            dateString: "$FECHA_REPORTE",
            format: "%Y-%m-%dT%H:%M:%SZ"
        }
    }
}
```

Listing 10: Cambio a date de las fechas

Todos los atributos que han necesitado un cambio al tipo *date* son:

CSV	Atributo
Áreas	FECHA_INSTALACION
Juego	FECHA_INSTALACION
Encuestas Satisfaccion	FECHA
Incidentes Seguridad	FECHA_REPORTE
meteo24	FECHA
Mantenimiento	FECHA_INTERVENCION
Incidencias Usuarios	FECHA_REPORTE

Cuadro 2: Atributos cambiados al tipo *date*

2.4.3. Paso a array de strings

El último paso del preprocesado en mongo pasa por la conversión de dos atributos específicos, *MantenimientoID* y *UsuarioID*, ambos pertenecientes a la colección *Incidencias Usuarios*. Ambos atributos se insertan inicialmente en mongo como un string con la siguiente forma: "['valor1', 'valor2']". Estos atributos son mal interpretados por mongo, ya que deberían ser arrays de strings, en vez de strings. Por este motivo, ambos atributos son preprocesados con el siguiente código:

```
$addFields: {
    Columna: {
        $split: [
            {
                $replaceAll: {
                    input: {
                        $replaceAll: {
                            input: {
                                $replaceAll: {
                                    input:
                                        {
                                            $replaceAll: {
                                                input: "$Columna",
                                                find: " ",
                                                replacement: ""
                                            }
                                        },
                                    find: "'",
                                    replacement: ""
                                }
                            },
                            find: "]",
                            replacement: ""
                        }
                    },
                    find: "[",
                    replacement: ""
```

```
            }
        }, ","]
    },
```

El código anterior simplemente reemplaza la columna por una nueva que está bien preprocesada. Este cambio de tipo se consigue a través del reemplazo de todos los valores problemáticos, como los espacios en blanco, las dobles comillas y los corchetes. Luego, este valor procesado se pasa a un operador *split* que se encarga de devolver un array de strings.

3. Validación de Esquemas

Para la validación de esquemas, se han desarrollado unos esquemas para cada csv que ha sido limpiado en el apartado anterior. Para ello, todos han seguido una estructura similar a la siguiente:

```
db.createCollection("nombre_de_colección", {
    validator: {
        $jsonSchema: {
            bsonType: "object",
            title: "Nombre_de_colección Validation",
            required: ["param1", "param2", "paramN"],
            properties: {
                param1: {
                    bsonType: "tipo1",
                    description: "descripción1"
                },
                param2: {
                    bsonType: "tipo2",
                    description: "descripción2"
                },
                paramN: {
                    bsonType: "tipo3",
                    description: "descripción3"
                }
            }
        }
    }
})
```

Listing 11: Estructura genérica de la validación de esquemas

Hemos aplicado un esquema adaptado a cada uno de los documentos con el fin de asegurar que cada dato tiene la estructura y tipos adecuados. Para ello se ha seguido el modelo porporcionado en el enunciado. Destacan el caso de *_id*, que se han decidido mantener como entero para la mayoría de las tablas ya que este tipo permite la generación de *UUIDs* de forma nativa desde `Mongo`. Esto facilitará futuras inserciones. Las únicas tablas que necesitan mantener el *_id* como string son *Usuarios* y *Mantenimiento*.

4. Creación de Agregados

Dada la propuesta de diseño de agregados se han relacionado las tablas originales para conseguir los agregados indicados, además de generar los atributos derivados con los datos originales.

4.1. Agregado de Area Recreativa Clima

4.1.1. Relaciones de Tablas con Áreas

Se han relacionado la tabla *Área* con las tablas de *juegos*, *incidentes de seguridad*, *encuestas satisfacción*, *estaciones_meteo_codigo_postal* y *meteo24*. Estas relaciones se han conseguido de la siguiente manera:

```
{
    $lookup: {
        from: '<tabla_a_relacionar>',
        localField: '<atributo_local_a_relacionar>',
        foreignField: '<atributo_de_la_tabla_a_relacionar>',
        as: '<atributo_resultante>'
    }
}
```

Listing 12: Operadores para relacionar tablas en mongodb

Es importante destacar que al final de la *pipeline* se transformarán las relaciones de *Juegos*, *Encuestas Satisifacción* y *estaciones_mete_codigo_postal* en tan solo su id a través de un proyecto, y para el caso de la relación con la tabla *Incidentes de Seguridad* se mostrarán atributos adicionales al id para completar el resumen.

4.1.2. Atributos Derivados

En el agregado *Area Recreativa Clima* se piden calcular los siguientes atributos:

- **estadoGlobalArea**

  Este atributo debe de contener el resultado de un cálculo que utilice la cantidad de incidentes de seguridad que tiene un área, la cantidad de juegos en reparación que tiene dicha área y la satisfacción de los usuarios en las encuestas. Para su creación se han seguido los siguientes pasos

  1. Cálculo de la nota de satisfacción

```
{
    $addFields: {
        encuestas_accesibilidad_transformado: {
            $map: {
                input: "$ref_encuestas_satisfaccion.
                   PUNTUACION_ACCESIBILIDAD",
                as: "puntuacion",
                in: { $subtract: [6, "$$puntuacion"] }
            }
        },
        encuestas_calidad_transformado: {
            $map: {
                input: "$ref_encuestas_satisfaccion.
                   PUNTUACION_CALIDAD",
                as: "puntuacion",
                in: { $subtract: [6, "$$puntuacion"] }
            }
        },
        nota_encuestas_area: {
            $sum: {
```

```
                $concatArrays: ["
                    $encuestas_accesibilidad_transformado", "
                    $encuestas_calidad_transformado"]
            }
        }
    }
}
```

Listing 13: Calculo del campo nota_encuestas_areas

2. Cálculo del número de incidentes de seguridad

```
numero_incidencias_ponderado: {
    $multiply: [{ $size: "$ref_incidentes_seguridad" }, 3]
}
```

Listing 14: Ponderación del número de incidencias

Se calcula el numero de incidentes de seguridad y se multplica por 3 para darle más peso en el cálculo.

3. Cálculo del número de juegos en reparación

```
juegos_mantenimiento: {
    $size: {
        $filter: {
            input: "$ref_juegos",
            as: "juego",
            cond: {
                $eq: ["$$juego.ESTADO", "EN REPARACION"]
            }
        }
    }
}
```

Listing 15: Calculo del número de juegos en reparación

Para conocer la cantidad de juegos en reparación de las distintas áreas se utiliza `$size` para conecer la lóngitud del array de devuelve el operador `$filter` tras filtrar por solo los juegos en reparación.

4. Suma de todos los valores calculados

```
$addFields: {
nota_total_area: {
    $sum:
        ["$nota_encuestas_area", "
            $numero_incidencias_ponderado", "
            $juegos_mantenimiento"]

    }
}
```

Listing 16: Calculo de not_total_area

5. Encontrar el valor máximo

```
{
    $group: {
        _id: null,
        max_nota_global: { $max: "$nota_total_area" },
        areas: { $push: "$$ROOT" }
    }
},
```

```
{
    $unwind: "$areas"
},
{
    $replaceRoot: {
        newRoot: {
            $mergeObjects: ["$areas", { max_nota: "
               $max_nota_global" }]
        }
    }
}
```

Listing 17: Agrupación de áreas previa a la normalización

Se busca obtener la nota máxima total de los juegos de todas las areas para estandarizar el atributo final

6. Cálculo final del atributo **estadoGlobalArea**

```
estado_global_area: {
    $round: [
        {
            $multiply: [
                { $divide: ["$nota_total_area", "$max_nota"]
                    },
                10
            ]
        },
        2
    ]
},
estado_global_area: {
                $subtract: [10, "$estado_global_area"]
}
```

Listing 18: Cálculo de estado_global_area

Dividimos la nota total de cada area por la nota máxima calculada en el paso anterior para estandarizar el valor, lo que se nos permite obtener un valor entre 0 y 1. Este valor se multiplca por 10 para obtener una nota, y se redondea a 2 decimales.

- **cantidadJuegosPorTipo**

Este atributo debe de contener el número de juegos que tiene cada área según al tipo al que correspondan. Para su obtención se han seguido los siguientes pasos:

1. Relacionamos los documentos de juegos y areas a través del campo *Area*. Además se separa el *array* de juegos en diferentes documentos para poder realizar las agrupaciones.

```
{
    // Areas con juegos
    $lookup: {
        from: 'juegos',
        localField: '_id',
        foreignField: 'AREA',
        as: 'aux_ref_juegos'
    }
},
{ $unwind: "$aux_ref_juegos" },
```

Listing 19: Relación de juegos y áreas

2. Agrupamos los juegos por tipo. Esto lo hacemos a través de 2 operaciones `$group`.

```
{
    $group: {
        _id: { tipo: "$aux_ref_juegos.tipo_juego", area_id:
            "$_id" }, // agrupar areas y tipos
        count: { $sum: 1 },
        ref_juegos: { $push: { _id: "$aux_ref_juegos._id" }
            },
        original: { $first: "$$ROOT" }
    }
},
{
    $group: { // Crear para cada área un array de {tipo,
        count}, {tipo, count}
        _id: "$_id.area_id",
        cuenta: {
            $push: {
                k: "$_id.tipo", v: "$count"
            }
        },
        ref_juegos: { $push: "$ref_juegos" },
        original: { $first: "$original" }
    },
}
```

Listing 20: Agrupacion de juegos por tipo

3. Creación del nuevo campo e inclusión en el documento original.

```
{
    $addFields: {
        "cantidad_juego_por_tipo": {
            $arrayToObject: "$cuenta"

        },
        "ref_juegos": {
            $reduce: {
                input: "$ref_juegos",
                initialValue: [],
                in: { $concatArrays: ["$$value", "$$this"] }
            }
        }
    }
},
{
    $replaceRoot: {
        newRoot: {
            $mergeObjects: [
                "$original",
                { "cantidad_juego_por_tipo": "
                    $cantidad_juego_por_tipo" },
                { "ref_juegos": "$ref_juegos" }
            ]
        }
    }
},
```

Listing 21: Creación de nuevo campo en el documento original

- **capacidadMax**

Este atributo debe de ser la cantidad máxima de juegos que un área es capaz de soportar. Para ello simplemente tomamos como máxima capacidad el número de juegos actual del área.

```
$addFields:{
    capacidadMax: "$TOTAL_ELEM"
}
```

Listing 22: Cálculo de la capacidad máxima

4.2. Agregado de Juegos

4.2.1. Relaciones de Tablas con Juegos

Se han relacionado con la tabla de *juegos* las tablas *incidencias de usuarios* y *mantenimiento*. Esto se ha conseguido al igual que en la anterior sección a través del operador `$lookup`.

Al final de la *pipeline* se transformarán las relación con *Mantenimiento* para solo mostrar sus id's y para el caso de la relación con *Incidencia* se mostrarán, además de sus id's, aquellos atributos que conformen el resumen.

4.2.2. Atributos Derivados

En el agregado *Juegos* se pide calcular los siguientes:

- **indicadorExposicion**

  Este atributo debe de ser un enumerado que corresponda con el nivel de exposición a los factores climatológicos del juego en concreto. Para ello se asignará a cada juego un valor aleatorio entre los valores 1, 2 y 3.

```
$addFields: {
    "indicadorExposicion": {
        $add: [
            {
                $floor: {
                    $multiply: [
                        { $rand: {} },
                        3
                    ]
                }
            },
            1
        ]
    }
}
```

Listing 23: Cálculo de indicadorExposicion

- **desgasteAcumulado**

  Este atributo hará uso del anterior atributo **indicadorExposicion** para calcular el nivel de desgaste de cada juego. Para ello se aplicará la siguiente fórmula: $(tiempo\_de\_uso \cdot indicadorExposicion) - (numero\_de\_mantenimientos \cdot 100)$

```
{
    $addFields: {
        "desgasteAcumulado": {
            $max: [
                {
                    $subtract: [
                        {
                            $multiply: [
                                {
```

```
                                        $add: [
                                            {
                                                $floor: {
                                                    $multiply: [
                                                        { $rand: {}
                                                            },
                                                        15
                                                    ]
                                                }
                                            },
                                            1
                                        ]
                                    },
                                    "$indicadorExposicion"
                                ]
                            },
                            {
                                $multiply: [
                                    { $size: "$ref_mantenimiento"
                                       },
                                    5
                                ]
                            }
                        ]
                    },
                    0
                ]
            }
        }
}
```

Listing 24: Cálculo de desgasteAcumulado

- **ultimaFechaMantenimiento**

  Este atributo debe de registrar la última fecha en la que se realizó un interveción en dicho juego. Para ello se toma la fecha máxima para cada juego.

```
$addFields: {
    "ultimaFechaMantenimiento": {
        $max: {
            $map: {
                input: "$ref_mantenimiento",
                as: "ref",
                in: "$$ref.FECHA_INTERVENCION"
            }
        }
    }
}
```

Listing 25: Cálculo de ultimaFechaMantenimiento

- **tiempoResolucion**

  Este atributo debe ser un valor entero que representa el tiempo que ha durado el proceso de mantenimiento. Consideramos que este atributo no debe de ubicarse en los juegos, dado que existe un tiempo de resolución de la incidencia para cada una de las incidencias, por lo que lo ubicaremos en el resumen que contiene juego para cada una de las incidencias.

  Para la creación de este atributo se han seguido los siguientes pasos:

1. Mientras se crea el resumen, en el cual se itera por cada incidencia, se calcula el atributo derivado

```
"res_incidencias_usuarios": {
    $map: {
        input: "$res_incidencias_usuarios",
        as: "ref",
        in: {
            ID: "$$ref._id",
            TIPO_INCIDENCIA: "$$ref.TIPO_INCIDENCIA",
            FECHA_REPORTE: "$$ref.FECHA_REPORTE",
            ESTADO: "$$ref.ESTADO",
            tiempoResolucion: {...}
        }
    }
}
```

Listing 26: Iterar por cada incidencia durante la creación del resumen

2. Se toma el máximo de todas las duraciones calculadas a través de la diferencia entre la fecha de la incidencia y las distintas fechas de cada uno de los mantenimientos.

```
$max: {
        $map: {
            input: "$$ref.MantenimientoID",
            as: "mantenimiento_id",
            in: {
                $subtract: [
                    { ... },
                    {
                        $$ref.FECHA_REPORTE
                    }
                ]
            }
        }
    }
```

Listing 27: Máximo de las duraciones calculadas

3. Finalmente se necesita obtener las fechas de los mantenimientos a los que se referencia en la incidencia.

```
$arrayElemAt: [
                {
                    $map: {
                        input: {
                            $filter: {
                                input: "$ref_mantenimiento",
                                as: "mantenimiento",
                                cond: { $eq: ["
                                    $$mantenimiento._id", "
                                    $$mantenimiento_id"] }
                            }
                        },
                        as: "man",
                        in: {
                            "$$man.FECHA_INTERVENCION"
                        }
                    }
                },
                0
```

```
            ]
```

Listing 28: Obtención de fechas de mantenimiento relacionadas con la incidencia

4.3. Agregado de Incidencias

4.3.1. Relaciones de Tablas con Incidencias

Este agregado relaciona la tabla de *Incidencias de Usuario* con la tabla de *Usuarios*. Para ello como para el resto de agregados se utilizará el operador `$lookup`.

En este caso no será necesario modificar los atributos que se muestran de la relación con *Usuarios* dado que la relación es embebida.

4.3.2. Creación de Atributo Derivados

En el agregado *Incidencia* se pide calcular el siguiente atributo:

- **nivelEscalamiento**

  Este atributo debe de categorizar la urgencia de las incidencias. Para ello se generará un número aleatorio del 1 al 10 que corresponderá a este valor.

```
$addFields: {
    nivelEscalamiento: {
        $add: [
            { $floor: { $multiply: [{ $rand: {} }, 10] } },
            1
        ]
    }
}
```

Listing 29: Cálculo de nivelEscalamiento

5. Conclusiones

Durante esta parte de la práctica nos hemos enfrentado a distintas problemáticas como son la limpieza de los datos de manera efectiva y el uso de `MongoDB` para la transformación de los datos y la construcción de agregados. Consumiendo estas partes el mayor tiempo de desarrollo de la práctica.

La limpieza de los datos ha sido, sin lugar a dudas, la parte más costosa en términos de tiempo. Los documentos csv contaban con numerosas instancias mal formateadas, gran cantidad de valores faltantes y campos muy poco explicativos. Esto ha llevado a numerosas iteraciones sobre el código para poder corregir correctamente los datos, además de dificultades en la relación de documentos como áreas y juegos; ya que estos no contaban con ningún campo que los relacionase explicitamente. Por otro lado, la imputación de valores faltantes como cadenas de caracteres ha generado más de una problemática durante la migración `Mongo`.

En cuanto a la migración, la principal dificultad ha sido la poca familiaridad con la operatividad de `MongoDB`. Pese a su parecido con lenguajes de programación tradicionales como `JavaScript` el uso de agregados incentiva un sobre anidamiento de operaciones que dificulta la legibilidad del código.

Con todo lo anterior, concluímos que esta práctica nos ha permitido experimentar y familiarizarnos con el entorno de `MongoDB`.