Arquitectura de Datos

PO2.2: MIGRACIÓN A MONGODB DE ACUERDO A CASOS DE USO

Curso 2025/2026

Grupo de prácticas: 10

Nombre	NIA	Correo Electrónico	Grupo
Javier Rosales Lozano	100495802	100495802@alumnos.uc3m.es	81
Alonso Rios Guerra	100495821	100495821@alumnos.uc3m.es	81
Nahuel Sebastián Vargas	100495715	100495715@alumnos.uc3m.es	81
Alejandro Rodríguez García	100495745	100495745@alumnos.uc3m.es	81

Fecha de entrega: 10/11/2025

Índice

1. Introducción

La siguiente memoria documenta el proceso de desarrollo de la segunda parte de la segunda práctica correspondiente a la asignatura Arquitectura de Datos del curso 2025/26. La práctica se basa en un contexto de diseño de un servicio de suscripción de televisión bajo demanda (películas y series).

En esta parte de la práctica se trabajará sobre un dataset de facturas que contiene inconsistencias y datos sucios. El objetivo es extraer, transformar y cargar los datos en MongoDB, así como explotarlos mediante consultas de agregación que den respuesta a distintos casos de uso definidos.

El dataset se proporciona a partir de diferentes ficheros como colecciones de documentos de facturas, donde cada registro representa la factura mensual de un cliente e incluye información relativa a tarifas, consumos y contrato; datos del cliente, e información embebida sobre los contenidos consumidos junto con estadísticas de uso.

La realización de la práctica se organiza según los siguientes procedimientos, atendiendo al enunciado de la práctica:

- **Limpieza y normalización de datos**: identificación y corrección de problemas de calidad en el dataset a partir de comandos y pipelines en MongoDB.
- **Reestructuración del modelo de datos**: transformación de la estructura inicial de la base y generación de nuevas colecciones separadas para películas y series/temporadas.
- **Validación de esquemas**: definición y programación de los esquemas de validación para cada colección creada para asegurar que los datos cumplan las reglas de consistencia y tipado especificadas.
- **Consultas de agregación**: implementación de los pipelines de agregación para resolver los casos de uso referenciados en la primera parte de la práctica (realizada anteriormente).

2. Normalización y limpieza de datos

Esta sección corresponde al contenido del primer fichero de texto entregado (**`PO22_81_10_1_limpieza.txt`**), dedicado a la normalización y limpieza de los datos del proyecto. El objetivo principal es garantizar la integridad, coherencia y calidad de los datos para un correcto análisis y uso posterior.

2.1. Presencia de identificadores duplicados o nulos

Antes de abordar la normalización, se realizó una comprobación sobre la **presencia de identificadores duplicados o nulos** en el dataset. Este paso es fundamental para evitar inconsistencias y posibles errores en las operaciones posteriores. Para ello, se ejecutaron dos consultas que verifican:

- La inexistencia de registros con identificadores **`null`** o **`undefined`**.
- La ausencia de duplicados en los campos identificadores.

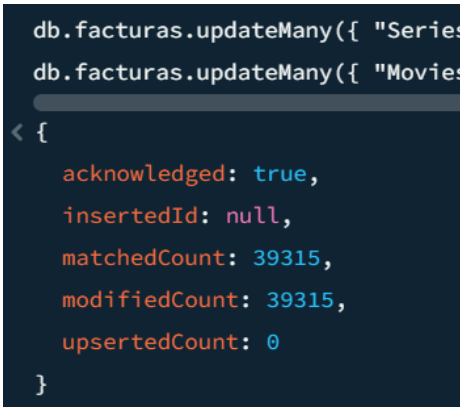
Los resultados confirman la ausencia de dichos problemas, ya que no hubo resultados visibles en pantalla tras la ejecución de esta parte.

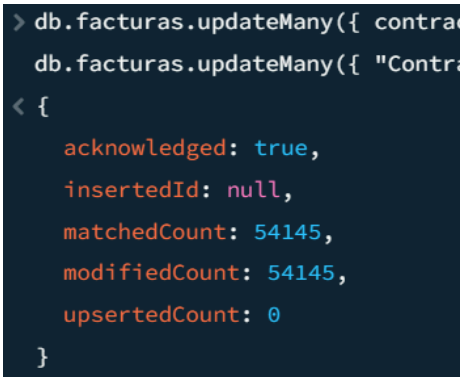
2.2. Normalización de los nombres de atributos

Una vez validada la calidad básica del dataset, se inició el proceso de homogeneización en la **nomenclatura de los atributos**. Se observó que una gran mayoría de los campos en los diferentes documentos y colecciones carecían de un formato común, lo que dificultaba la manipulación y consulta homogénea de los datos. Para solventarlo, se establecieron las siguientes reglas para la estandarización de nombres:

- Los atributos que representan **campos simples** (es decir, que no corresponden a objetos anidados ni colecciones dentro de un documento) se formatean en `snake_case`. Por ejemplo, un campo descrito como "end date" pasa a llamarse "end_date". Esta convención facilita la lectura y el uso constante de los nombres en scripts y consultas.
- Los atributos que representan **objetos complejos o arrays de objetos** mantienen una convención con la primera letra en mayúscula, para diferenciar claramente los documentos embebidos o colecciones anidadas; como Cliente, Producto, etc.

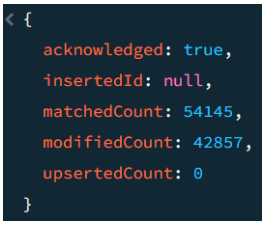
Se empleó **`updateMany()`** en MongoDB para actualizar únicamente los documentos que contenían el atributo, orientando la actualización solo a aquellos documentos en los que el atributo en cuestión existiera, garantizando seguridad y eficiencia. Las operaciones se realizaron por separado para Movies y Series, puesto que dichos atributos no están presentes en todas las instancias.





2.3. Revisión y corrección de tipos de datos

A continuación, se realizó un análisis exhaustivo sobre el **tipo de dato correspondiente a cada atributo** de acuerdo con el enunciado de la práctica. Esto es crucial para asegurar que la base de datos utilice los formatos correctos y optimizados para cada campo, permitiendo desde cálculos monetarios hasta comparaciones temporales eficientes. A modo de resumen, la clasificación general fue la siguiente:



Tipo de dato	Formato	Atributos
Fechas y horas; momentos temporales	`ISODate`	billing, charge_date, dump_date, Client.birth_date, Contract.end_date, Contract.start_date, Movies.date_time, Movies.License.date_time, Series.date_time, Series.License.date_time
Importes monetarios	`Decimal128`	TOTAL, Contract.Product.cost_per_content, Contract.Product.cost_per_day, Contract.Product.cost_per_minute, Contract.Product.cost_per_view, Contract.Product.monthly_fee, Movies.Details.budget, Movies.Details.gross
Códigos e identificadores	`String`	_id, Client.customer_code, Client.dni, Contract.contract_id
Referencias a documentos de otras colecciones	`ObjectId`	Client, Contract, Contract.Product
Nombres, títulos y descripciones	`String` **(homogeneizar mayúsculas y tildes)**	Client.email, Client.name, Contract.address, Contract.country, Contract.town, Contract.Product.reference, Contract.Product.type, Movies.title, Movies.Details.color, Movies.Details.content_rating, Movies.Details.country, Movies.Details.imdb_link, Movies.Details.language, Movies.Details.Cast.Stars.player, Movies.Details.Director.name, Series.title
Listas de valores	`Array<String>`	Client.surname, Movies.Details.genres, Movies.Details.keywords
Porcentajes o métricas de consumo	`Int32` **(0-100) o** `Double` **(según la precisión)**	Contract.Product.zapping, Movies.Details.aspect_ratio, Movies.Details.imdb_score, Movies.viewing_pct, Series.viewing_pct
Duración, contadores y cantidades enteras	`Int32`	Client.phone, Contract.zip, Contract.Product.promotion, Movies.Details.critic_reviews, Movies.Details.duration, Movies.Details.facebook_likes, Movies.Details.faces_in_poster, Movies.Details.user_reviews, Movies.Details.voted_users, Movies.Details.year, Movies.Details.Cast.facebook_likes, Movies.Details.Cast.Stars.facebook_likes, Movies.Details.Director.facebook_likes, Series.avg_duration, Series.episode, Series.season, Series.total_episodes, Series.total_seasons

2.4. Tratamiento de valores no definidos (undefined)

Durante la limpieza del dataset, se detectó la **presencia de valores `undefined`**, derivados de inconsistencias en la ingestión o ausencia estructural de datos. En MongoDB, **`undefined` no es equivalente a `null`** ni a un campo inexistente, lo que puede producir errores en agregaciones, comparaciones y procesos analíticos. Por ello, se estableció una **normalización basada en convertir cualquier valor no definido en `null` explícito** para no alterar el conteo y proporción de los datos.

El tratamiento se abordó según la complejidad estructural del dato:

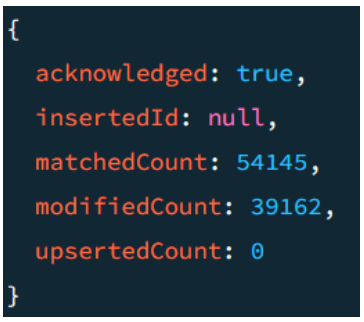
- **Campos simples u objetos embebidos**: se empleó `updateMany()` con `$ifNull` para reemplazar `undefined` por `null` de forma directa y eficiente, sin alterar valores ya existentes.
- **Arrays de objetos**: se utilizaron `$map` y `$mergeObjects` con lógica `$cond` + `$ifNull` para asegurar la presencia del atributo en cada elemento, garantizando integridad estructural, aunque con mayor coste en arrays grandes.

La muestra de los resultados por pantalla garantiza que se han modificado los datos correctamente.

2.5. Homogeneización de tildes

Finalmente, para conseguir uniformidad en las cadenas de texto, se procedió a la **homogeneización de los valores de los campos tipo `string`**. En concreto:

- Se optó por **convertir a mayúsculas** todos los campos tipo string **con excepción del correo electrónico** (Client.email), **que se dejó en minúsculas** para respetar convenciones y evitar confusiones en búsquedas o validación de emails.
- De forma paralela, se corrigieron las letras con tildes o acentos, sustituyendo estos caracteres por sus **equivalentes sin símbolo** (ejemplo: "á" a "a"). Esto previene problemas en consultas, comparaciones y agrupamientos con textos acentuados.



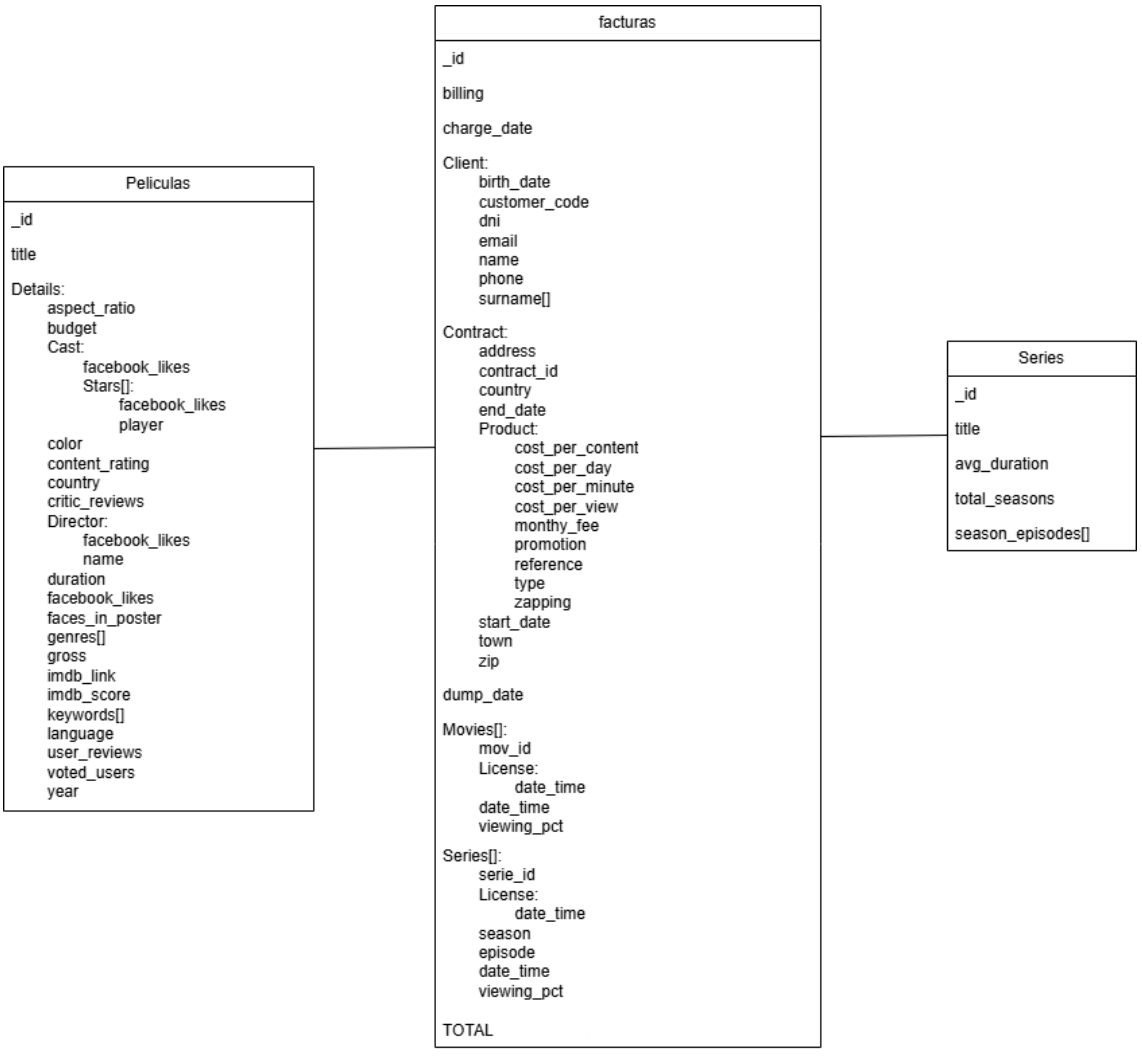
Esta transformación se implementó mediante pipelines de `updateMany()` que iteran sobre los atributos indicados, llevando a cabo el reemplazo de caracteres y la conversión de mayúsculas, asegurando la coherencia y normalización de los datos a nivel global.

3. Reestructuración del modelo de datos

La siguiente sección corresponde al contenido del segundo fichero de texto (**`P022_81_10_2_reestructuracion.txt`**), dedicado a la reestructuración del modelo de datos original. Para este apartado del proyecto, se deben migrar los datos originales limpiados anteriormente a una nueva base de datos modelizada con tres documentos distintos: facturas, películas y series.

3.1. Estructura actualizada de la base de datos

Tal y como se pedía en la práctica, se ha actualizado la estructura de la base de datos, creando las **dos colecciones** nuevas mencionadas anteriormente. A continuación, se observa un **diagrama UML** con el modelo de datos completo:



Implementar esta nueva estructura nos aportará mejoras en **eficiencia**, al no tener todos los datos embebidos, además de que nos permitirá **aprovechar los datos** para ofrecer métricas de consumo a clientes y **segmentar la venta de publicidad en plataformas**.

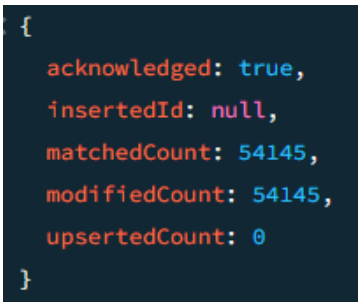
3.2. Creación de la colección de Películas

Antes de realizar la separación de los datos, realizamos un **rápido análisis** de los datos relacionados con las películas. Tras este análisis, llegamos a la conclusión de que en facturas debían guardarse datos como ID de la película, licencia del cliente para la película, fecha y hora de visionado y el porcentaje de visualización de la película. Estos datos serán almacenados en cada factura dentro del array Movies[], donde cada posición almacenará los datos anteriores para cada película.

Por otro lado, en la nueva colección de Peliculas almacenaremos el resto de datos referentes a la película. Principalmente almacenaremos datos como ID, título y detalles. Para crear el **ID** para cada película decidimos usar una parte del imdb_link (ej: HTTP://WWW.IMDB.COM/TITLE/**TT0056172**/?REF_=FN_TT_TT_1 , de donde extraemos el ID: **TT0056172**.

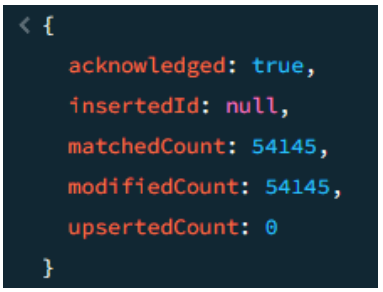
Para lograr esta división seguimos la siguiente **metodología**:

1. Primero generamos un **atributo temporal (movies_ref)** en facturas donde almacenamos datos como el ID de la película, licencia del cliente para la película, fecha y hora de visionado y el porcentaje de visualización de la película. Obteniendo 54145 cambios, todos los registros, como era de esperar.



2. Usamos un **pipeline de agregación** para crear la nueva colección de Peliculas, donde usamos comandos como **$unwind** para realizar esa separación. En la nueva colección almacenamos datos como el ID de la película, el título y sus detalles.

3. Por último, realizamos **operaciones menores** como eliminar el campo Movies original de facturas, copiar los datos almacenados en el atributo temporal (movies_ref) al array Movies y como paso final, eliminar el campo temporal movies_ref. El resultado por pantalla de estas operaciones fue el mismo. Se actualizaron todos los campos, como era de esperar.



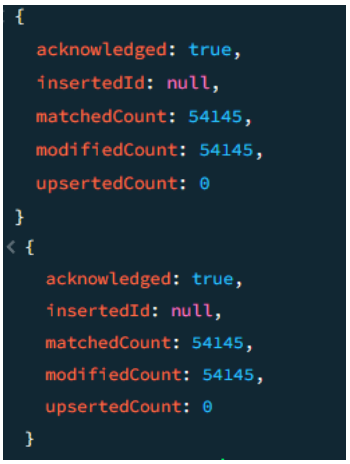
3.3. Creación de la colección de Series

Antes de realizar la separación de los datos, realizamos un **rápido análisis** de los datos relacionados con las series. Tras este análisis, llegamos a la conclusión de que en facturas debían guardarse datos como ID de la serie, licencia del cliente para la serie, fecha y hora de visionado, el porcentaje de visualización de la serie, la temporada y el episodio de la serie. Estos datos serán almacenados en cada factura dentro del array Series[], donde cada posición almacenará los datos anteriores para cada episodio de una serie.

Por otro lado, en la nueva colección Series almacenaremos el resto de datos referentes a la película. Principalmente almacenaremos datos como el ID de la serie, título, duración media del episodio, número de temporadas y un array seasons_episodes con el número de episodios por temporada y posición. Para crear el **ID** de cada serie optamos por hacer la siguiente concatenación: **TITLE/TOTAL_SEASONS** (ej: VIKINGS/4).

Para lograr esta división seguimos la siguiente **metodología**, idéntica a la de las películas:

1. Primero generamos un atributo temporal (series_ref) en facturas donde almacenamos datos como el ID de la serie, licencia del cliente para la serie, fecha y hora de visionado, el porcentaje de visualización de la serie, la temporada y el episodio de la serie. Obteniendo 54145 cambios, todos los registros, como era de esperar.



2. Usamos un **pipeline de agregación** para crear la nueva colección de Series, donde usamos comandos como **$unwind** para realizar esa separación. En la nueva colección almacenamos datos como el ID de la serie, título, duración media del episodio, número de temporadas y un array seasons_episodes con el número de episodios por temporada y posición.

3. Por último realizamos **operaciones menores** como eliminar el campo Series original de facturas, copiar los datos almacenados en el atributo temporal (series_ref) al array Series y como paso final, eliminar el campo temporal series_ref. El resultado por pantalla de estas operaciones fue el mismo. Se actualizaron todos los campos, como era de esperar.

4. Validación de esquemas

Pasamos a la **validación de esquemas del nuevo diseño de la base de datos NoSQL**, una fase fundamental para garantizar la integridad, consistencia y calidad de los datos nuevos en su inserción, e implementada en el tercer fichero de la entrega final (**`PO22_81_10_3_esquemas.txt`**). Debido a que la creación de los agregados no garantiza un tipo de dato fijo para cada atributo, es necesaria esta parte para no tener que limpiar constantemente los datos.

Durante la realización de esta sección, se ha procurado tener un riguroso cuidado para definir y aplicar validadores de esquema que aseguren que cada documento en las colecciones principales cumpla con las normas estructurales y de tipo establecidas. Para ello, se ha utilizado la **tabla de tipos de datos** definida en anteriormente en la limpieza de datos, y la herramienta Schema proporcionada por la propia aplicación **MongoDB Compass**. Es importante destacar que, por simplicidad, seguridad de los datos y porque no se especificaba, **se marcaron como obligatorios todos los campos de cada documento y subdocumento en las tres validaciones**. Para cada colección, se diseñó un esquema detallado que:

- Define los **tipos de todos los atributos** de los documentos, diferenciando entre objetos, arrays y tipos.
- Establece los **campos obligatorios** mediante la **cláusula `required`** para garantizar que la información imprescindible nunca falte.
- Aplica **expresiones regulares** con **`pattern`** para aquellos campos de tipo string que requieren un formato específico (todas las letras en mayúsculas salvo el atributo Client.email).
- Incluye **descripciones precisas y claras** para cada atributo, fomentando la documentación interna y facilitando el mantenimiento futuro.
- Considera **subesquemas para objetos anidados** y arreglos de documentos, con validaciones adicionales sobre sus campos internos para mantener la consistencia a todos los niveles.
- Configura **`validationLevel` de forma moderada (`moderate`)** para asegurar que la validación no sea estricta en todo momento, y **`validationAction` en error (`error`)** para evitar inserciones o actualizaciones con anomalías.

En líneas generales, para cada colección se ha desarrollado un esquema JSON que establece los tipos BSON esperados para cada campo, define los campos obligatorios mediante la cláusula `required`, y aplica expresiones regulares para validar formatos específicos en cadenas de texto. Estas definiciones están acompañadas por descripciones que mejoran la documentación interna y favorecen el mantenimiento futuro.

Aunque las validaciones comparten una base común, cada una se adapta a las particularidades del tipo de datos y modelo que maneja la colección específica, aplicando desde validaciones simples de tipo y patrón, hasta validación condicional sobre colecciones anidadas. Esto no solo previene errores en la inserción y actualización, sino que también **optimiza la base de datos para las consultas** que se realizarán en el siguiente apartado.

5. Consultas de agregación

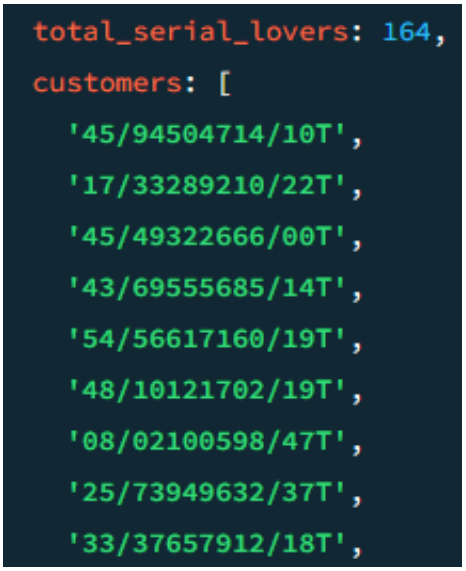
Finalmente, concluimos la práctica con el último apartado relacionado con el desarrollo de consultas de agregación relacionadas con el contexto propuesto en el proyecto. El código relacionado con este apartado de la memoria se asocia con el último fichero a entregar en la raíz del proyecto (**`PO22_81_10_3_agregaciones.txt`**). A continuación, se dedica un apartado para cada una de las consultas:

5.1. Q1: Serial Lovers

Para esta primera consulta, el objetivo consiste, tal como se pide, en **identificar todo aquel cliente que ha visto, al menos. una temporada completa de una serie**. Para ello se han realizado una serie de supuestos, entre los que se encuentran que, **para que un capítulo de una serie se considere como visto ha de tener un valor de porcentaje visto igual o superior al 95%**, valor contenido en viewing_pct e integrado de la siguiente manera:

```
“{ $match: { "Series.viewing_pct": { $gte: 95, $lte: 100 } } }”.
```

Además, se ha añadido un **contador de clientes únicos** de forma que se muestre por pantalla el número exacto de serial lovers que hay, para posteriormente mostrar el identificador de cada cliente.



A continuación, la agrupación por código de usuario, serie, temporada y episodio garantiza que **cada episodio visto por un cliente se cuente una sola vez**, incluso si lo ha visto varias veces. Después, se conecta con la colección Series para obtener información sobre la serie, concretamente, el número total de episodios en cada temporada (guardado en el array season_episodes).

Luego, **solo se seleccionan los clientes que hayan visto el número total de episodios de una temporada específica de una serie**; es decir, se confirma que el usuario vio todos los episodios únicos de alguna temporada.

Por último, se garantiza que **cada cliente aparezca solo una vez en el resultado**, obteniendo la lista única de "serial lovers", para después **devolver tanto el total de clientes que cumplen la condición como el listado de sus códigos únicos**. La imagen de la derecha muestra una parte resultante de la ejecución de esta consulta:

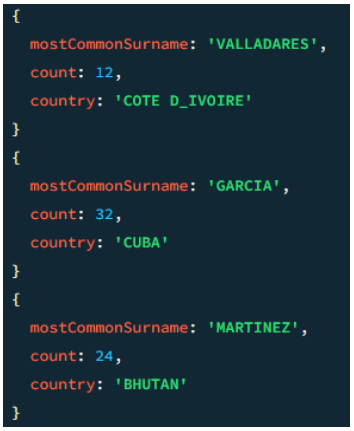
5.2. Q2: Apellidos comunes.

En esta segunda consulta se pide **obtener el apellido más frecuente por país de contrato**. Para ello, se descompone el array Client.surname para que cada apellido sea un documento independiente. Esto es necesario para poder contar la frecuencia de cada apellido individualmente, ya que originalmente los apellidos de un cliente están agrupados en un array:

```
{ $unwind: "$Client.surname" }
```

A continuación, se agrupan los documentos ya descompuestos por la combinación de país (Contract.country) y apellido (Client.surname), y se cuenta cuántas veces aparece cada combinación. Esto nos da la **frecuencia de cada apellido dentro de cada país**.

Después, los resultados se ordenan, primero, por país en orden alfabético ascendente, y dentro de cada país por la frecuencia (count) de los apellidos de forma descendente. Esto ordena los **apellidos más frecuentes para cada país al principio**. Se agrupan los resultados por país para quedarnos solo con el apellido que apareció primero tras el ordenamiento anterior. Debido a ese orden, **el apellido de mayor frecuencia queda seleccionado como mostCommonSurname**. Se captura también la cantidad o frecuencia con la que aparece.



Finalmente, se proyectan sólo los campos relevantes, sin el campo _id: se presentan **el país, el apellido más común y la frecuencia** para facilitar la interpretación y generación de informes.

5.3. Q3: Actores populares

Esta consulta **obtiene el listado con el top-5 de actores más vistos para clientes cuyo contrato es de España**.

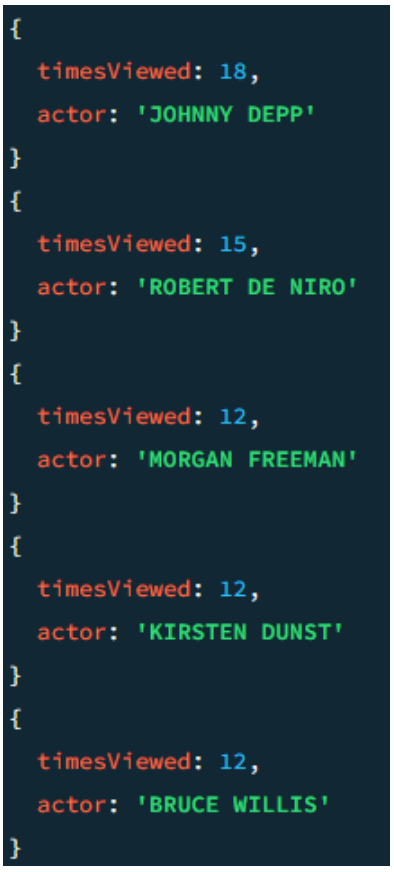
```
{ $match: { "Contract.country": "SPAIN" } }
```

Se utiliza `$unwind` para **transformar el array de películas vistas en documentos individuales**, permitiendo analizarlas por separado. Luego, se ejecuta un `$lookup` con la colección Peliculas para traer los detalles completos de cada película vista, especialmente el reparto de actores. `$unwind` descompone los detalles traídos para analizar cada película por separado.

La consulta extare el array de actores (Stars) en cada película, obteniendo **un documento por cada actor que ha participado en las películas vistas**.

```
{ $unwind: "$movieDetails.Details.Cast.Stars" }
```

Por último, **se agrupa por el nombre del actor y se suma el número de películas en que aparece** y son vistas por clientes españoles; después, ordena los actores por número de visualizaciones descendente y **selecciona los cinco actores más vistos** y muestra por pantalla los resultados:nombre del autor y la cantidad de visionados. De esta manera, podemos apreciar este orden de la forma correcta.



5.4. Q4: Clásicos modernos

Esta consulta permite obtener, **para cada año del siglo XX (1900-1999), el listado de las 10 películas más vistas** según el registro de visualizaciones en la base de datos.

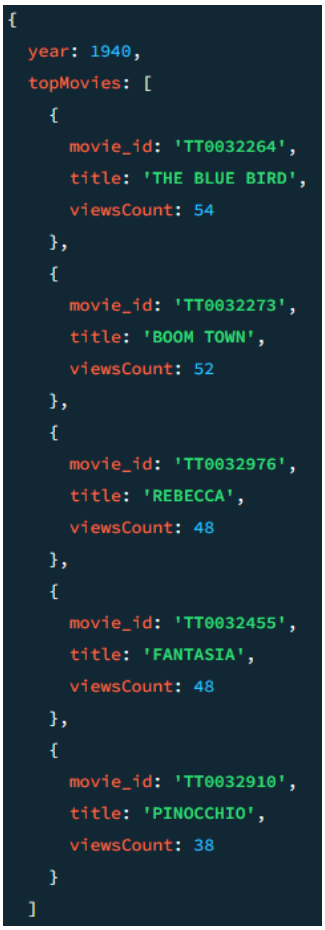
Puesto que se pide el top 10 anual en el siglo XX, **se ha interpretado que devuelva un top 10 de las películas por cada año**, devolviendo así el correspondiente top para todos los años del siglo XX. Para empezar, se deconstruye el array de películas en cada factura, generando **un documento por cada película vista** por cada cliente.

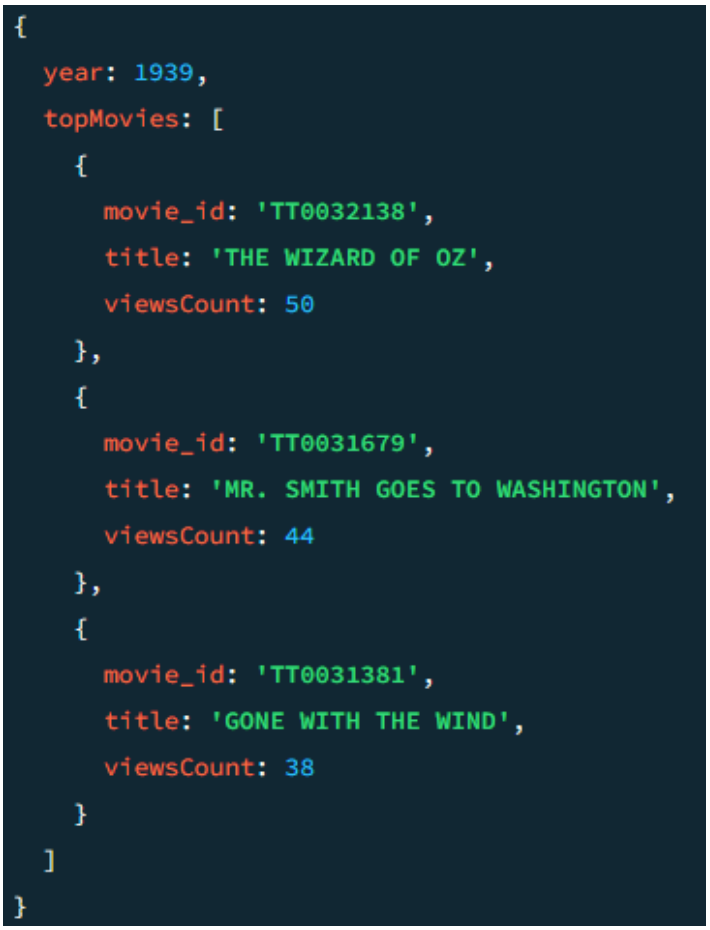
```
{ $unwind: "$Movies" }
```

A continuación, **se une cada registro de visualización de factura con su correspondiente película**, extrayendo así metadatos útiles como el año y el título y se mantiene únicamente aquellos registros que correspondan a películas producidas entre los años 1900 y 1999.

```
{ $match: {"movieDetails.Details.year": { $gte: 1900, $lte: 1999 } } }
```

Cada combinación año-película se agrupa y se contabiliza el número total de visionados de cada película anualmente, y dentro de cada año las películas se ordenan según su cantidad de vistas de mayor a menor. Finalmente, se juntan los resultados por año y **se seleccionan solo las primeras 10 películas más vistas para cada año** (top-10) y se ordena el resultado anual de forma cronológica para facilitar la consulta y el análisis.





5.5. Q5: Facturación mensual

Esta consulta calcula el **total de ingresos en un mes concreto**. En este caso, se ha elegido **febrero de 2016** (como podía haber sido cualquier otro) ya que la consulta a realizar no lo especifica en la memoria (“total de ingresos en un mes”).

Para comenzar, **se seleccionan únicamente las facturas cuya fecha de facturación (billing) esté dentro del mes** de febrero de 2016, incluyendo el primer día del mes y excluyendo el primer día del mes siguiente.

```
$gte: ISODate("2016-02-01T00:00:00Z"), $lt: ISODate("2016-03-01T00:00:00Z")
```

A continuación, todos los documentos que pasaron el filtro se agrupan en un único grupo ("_id: null"), y **se suman los valores del campo TOTAL** (importe de cada factura) para conocer el ingreso global del mes filtrado. Finalmente, **se muestra el resultado como un objeto plano con solo el total de ingresos (totalIngresos)**. A continuación se el resultado de la ejecución de la consulta:

```
{
  totalIngresos: Decimal128('390114.94000000000000')
}
```

5.6. Q6: Consumo mensual

Esta consulta obtiene el **consumo mensual total dividido entre películas y series, indicando tanto el número de visionados como la duración total en horas de cada mes**.

Primero, la consulta descompone los visionados de películas, extrae el mes y año de cada visionado, y cruza cada uno con la colección Peliculas para obtener la duración real. Se multiplica la duración por el porcentaje visto y lo convierte en horas, agrupando todo por mes y año para sumar la cantidad total de visualizaciones y horas consumidas por películas. De la misma manera, con el operador **`$unionWith`**, realizamos el mismo proceso para los episodios de series: descompone el array, identifica el mes y el año, y cruza con la colección Series para obtener la duración promedio (avg_duration). Se calculan las horas vistas ajustando por porcentaje de visionado, y vuelve a agrupar por mes y año para obtener el total de visionados y horas consumidas por series.

```
{
  visionados: 16989,
  horas: 24815.255,
  tipo: 'Pelicula',
  mes: 1,
  ano: 2016
}
{
  visionados: 25389,
  horas: 16848.6565,
  tipo: 'Serie',
  mes: 1,
  ano: 2016
}
```

Finalmente, **ambas métricas se combinan y se proyectan juntas**, de modo que **cada fila indica el consumo mensual para películas y series**, facilitando la comparación y el estudio de hábitos de usuario. A continuación se muestra una parte resultante de la ejecución de la consulta:

```
{
  visionados: 17346,
  horas: 25535.127,
  tipo: 'Pelicula',
  mes: 2,
  ano: 2016
}
{
  visionados: 25448,
  horas: 16800.800833333335,
  tipo: 'Serie',
  mes: 2,
  ano: 2016
}
```

5.7. Q7: Películas exitosas

Esta consulta obtiene el **número total de visionados y duración acumulada para una película concreta**, elegida por el equipo, añadiendo campos. En este caso, utilizamos la referencia **"TT0056172"**.

Para ello, se diseñó una consulta de agregación en MongoDB que permite extraer estos datos a partir de la colección facturas, que contiene registros de visualización de contenido por parte de los clientes.

La consulta desempaqueta el array de películas vistas en cada factura mediante el operador **`$unwind`**, seguido de un filtro que selecciona una película específica cuyo rendimiento se desea analizar. Posteriormente, **se realiza una operación `$lookup` para incorporar los datos de duración de la película desde la colección Peliculas**, asegurando el enlace correcto entre la referencia del identificador de la película en facturas y su detalle en la colección maestro.

Luego, **se calculan los minutos vistos** multiplicando el porcentaje de visualización por la duración total de la película, dividiéndolo entre 100; y finalmente, **se agrupan los resultados para obtener el total de visionados y la suma acumulada de minutos vistos**, presentando estos datos junto con el título de la película. Este diseño permite evaluar el éxito comercial y el grado de consumo efectivo de cada película específica.


```
{
  total_visionados: 46,
  minutos_acumulados: 8578.33,
  titulo: 'LAWRENCE OF ARABIA',
  pelicula_id: 'TT0056172'
}
```


5.8. Q8: Fracaso en series

Esta consulta tiene como objetivo evaluar el comportamiento de consumo de las series televisivas desde la perspectiva individual de cada usuario, centrándose en el **porcentaje promedio de visualización por serie y por usuario**. Para ello, se parte de la colección de facturas, en la cual cada documento contiene arrays embebidos de visualizaciones de series. El proceso inicia con la **separación de cada visualización individual** mediante el operador **`$unwind`** sobre el array de series, para posteriormente agrupar estos datos por usuario y por serie correspondiente. Esta agrupación permite **calcular el promedio del porcentaje de visionado realizado por cada usuario**, utilizando la función agregada **`$avg`**, mostrando así la tendencia media de consumo o abandono del contenido.


```
{
  media_pct: 100,
  total_vistos: 2,
  usuario: '40447572W',
  serie: 'THIRTYSOMETHING/4'
}
```

```
{
  media_pct: 63.375,
  total_vistos: 8,
  usuario: '65235144Z',
  serie: 'TWO AND A HALF MEN/12'
}
```

6. Declaración de uso de IA

Para el desarrollo de la práctica nos hemos apoyado en herramientas de IA generativa como Perplexity o ChatGPT. Estas fueron usadas para ayudarnos a entender conceptos, sintaxis de comandos y, ocasionalmente, ayudarnos a detectar errores en nuestro código. Para ello creamos un espacio de trabajo en modo estudio donde la IA basaba sus respuestas en las fuentes subidas como apuntes de clase y laboratorios.