Migración de un Modelo Relacional a Documental en Datos de Streaming

Joana Auriello, Pablo Molina, *Maestría en Ciencia de Datos y Aprendizaje Automático*
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
joana.auriello@gmail.com / molinafernandez.pablo@gmail.com

Resumen

Este trabajo presenta la migración de un modelo relacional a uno documental en MongoDB, aplicado a una base de datos de visualizaciones en plataformas de streaming. La transformación se realizó con Python y MongoDB Atlas, y se evaluaron tiempos de ejecución entre ambos modelos.

I. Introducción

Este trabajo final se centra en la migración de un modelo de base de datos relacional hacia un modelo documental utilizando MongoDB. La transformación se realizó sobre una base de datos real vinculada al dominio de plataformas de streaming, con el objetivo de mejorar la eficiencia de consultas y la escalabilidad del sistema.

El proyecto se enmarca en la familia de trabajos orientados a la migración entre modelos, elegida por representar un escenario realista y desafiante en el que se busca transformar una base de datos relacional tradicional en una solución más flexible y eficiente. La base original, conocida por los integrantes del equipo a partir de su experiencia profesional, gestiona información sobre visualizaciones de contenido audiovisual en apps de streaming, organizada mediante un esquema relacional clásico con una tabla de hechos central y múltiples tablas de dimensiones. Este tipo de modelo, si bien robusto, presenta limitaciones en términos de rendimiento cuando se trata de consultas analíticas complejas o de alto volumen. La migración hacia un modelo documental permite reducir la dependencia de joins, optimizar el acceso a datos relevantes y facilitar la escalabilidad de la solución, adaptándose mejor a las necesidades actuales de análisis y rendimiento.

El objetivo principal del trabajo fue migrar el modelo relacional de visualización de contenidos en plataformas de streaming a una estructura documental en MongoDB. Este proceso implicó analizar la estructura relacional existente, diseñar un nuevo modelo documental considerando estrategias adecuadas de embebido y referenciación, aplicar patrones como Extended Reference, y desarrollar consultas representativas para validar la eficiencia del nuevo modelo. Finalmente, se realizó una evaluación comparativa entre ambos enfoques para medir mejoras en rendimiento, simplicidad de consultas y adaptabilidad del sistema a futuros requerimientos.

II. Descripción de la Base de Datos Relacional Original

II-A. Tipo de datos e información contenida

La base de datos relacional utilizada en este trabajo se construye a partir de los registros (logs) generados por aplicaciones de servicios de streaming. Estos logs contienen información sobre el consumo de contenido por parte de usuarios finales e incluyen atributos vinculados a los dispositivos utilizados, sistemas operativos, plataformas de acceso, ubicaciones geográficas y otros elementos relacionados con el contexto de visualización. Los datos son procesados mediante pipelines ETL implementados en la nube, los cuales extraen, transforman y cargan la información de forma periódica en un data warehouse corporativo. Por razones de privacidad y volumen, en este trabajo se empleó una submuestra representativa de la base completa. La implementación se limitó a un único cliente (una aplicación de streaming específica) y se centró en un subconjunto del esquema general. La estructura relacional empleada se encuentra altamente normalizada.

El modelo se compone de una tabla de hechos principal (`fact_vod_details`) y múltiples tablas de dimensiones, cada una representando una entidad del dominio: cliente, dispositivo, ubicación, socio (partner), plataforma, fecha y contenido (media). Cada dimensión contiene su propia clave primaria y atributos descriptivos, mientras que la tabla de hechos almacena únicamente las llaves foráneas que referencian a estas dimensiones, junto con los campos de medida relevantes como el identificador del usuario (`uuid_hashvalue`), el identificador de sesión (`sessionid_hashvalue`) y los segundos de visualización (`seconds_elapsed`). Esta estructura busca minimizar la redundancia de datos y asegurar la integridad referencial, donde cada hecho puede asociarse de forma precisa con las entidades correspondientes mediante sus claves. El diagrama de la Figura 1 representa el esquema utilizado por la base de datos relacional.

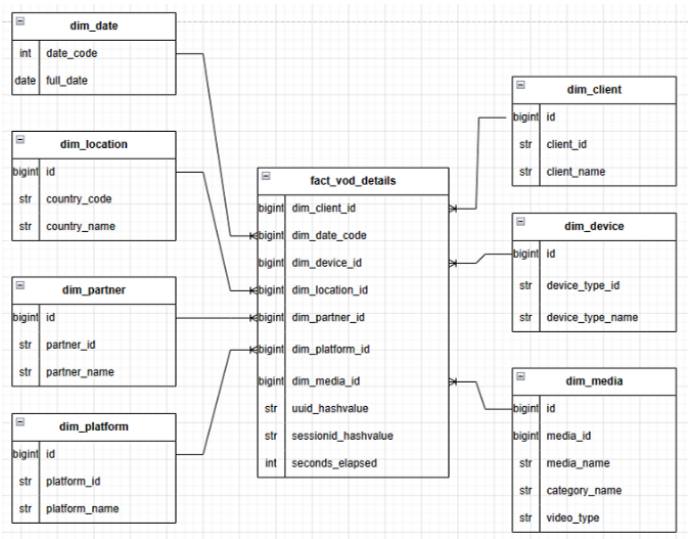

Figura 1: Diagrama relacional: base de datos de Streaming

El modelo relacional está estructurado en torno a una tabla de hechos denominada `fact_vod_details`, que registra eventos individuales de visualización de contenido. Esta tabla contiene los datos centrales del sistema, incluyendo campos clave como `uuid_hashvalue` (identificador anonimizado del usuario), `sessionid_hashvalue` (identificador único de sesión) y `seconds_elapsed` (cantidad de segundos de visualización por evento). Además, incluye múltiples llaves foráneas que se vinculan con sus respectivas tablas de dimensiones: `dim_date`, `dim_client`, `dim_device`, `dim_location`, `dim_partner`, `dim_platform` y `dim_media`. Cada tabla dimensional contiene los atributos necesarios para describir la entidad correspondiente y su business key, lo que garantiza una correcta integración de los datos.

Es importante destacar que los identificadores `uuid_hashvalue` y `sessionid_hashvalue` fueron diseñados para garantizar la anonimización de los usuarios. El primero representa un usuario individual y es generado a partir de una combinación de características del dispositivo utilizado (por ejemplo, tipo de hardware, sistema operativo), junto con atributos geográficos y la dirección IP. Por su parte, el identificador de sesión se construye considerando también elementos temporales, de modo que cada vez que un usuario inicia una nueva sesión de visualización, se genera un valor único. Estos mecanismos permiten llevar un control granular y preciso sobre las interacciones de los usuarios con la plataforma sin comprometer su privacidad.

Este diseño permite mantener un repositorio detallado, normalizado y reutilizable, adecuado para análisis estructurados. No obstante, el grado de normalización, aunque útil para mantener integridad y evitar duplicaciones, también implica una dependencia significativa de operaciones de unión entre tablas (joins) para resolver consultas analíticas completas. Esta característica, en escenarios de alta demanda y volumen de datos, puede derivar en problemas de escalabilidad y tiempos de respuesta elevados, especialmente cuando se busca construir en tiempo real funcionalidades como rankings dinámicos de contenido o colecciones personalizadas dentro de las aplicaciones de streaming.

II-B. Periodo temporal analizado

El análisis se concentró en los primeros diez días del mes de mayo de 2025, incluyendo al menos un fin de semana, con el objetivo de capturar variaciones en el comportamiento de uso en días de mayor tráfico. Este rango fue elegido por su relevancia práctica, ya que permite evaluar el rendimiento del sistema en escenarios de alta demanda, especialmente en tareas críticas como la generación automática de listas de contenido popular dentro de la aplicación.

II-C. Limitaciones del modelo relacional en este caso de uso

Si bien el modelo relacional utilizado proporciona una estructura clara y normalizada, presenta limitaciones relevantes en escenarios de análisis intensivo y gran volumen de datos, como es el caso de las plataformas de streaming. Una de las principales dificultades es el tiempo de procesamiento requerido para ejecutar consultas analíticas complejas, que involucran múltiples uniones entre tablas (joins), filtrados y agregaciones. Estas operaciones, comunes en entornos donde se genera y analiza una gran cantidad de datos diariamente, tienden a ser costosas en términos de recursos computacionales y pueden impactar negativamente en los tiempos de respuesta.

Una situación concreta en la que estas limitaciones se manifiestan es la generación dinámica de colecciones dentro de las aplicaciones de streaming. Estas colecciones agrupan contenidos bajo ciertos criterios, por ejemplo, “lo más visto”, “favoritos” o “top 10”, y se actualizan constantemente para reflejar el comportamiento más reciente de los usuarios. Su construcción

depende de consultas frecuentes que filtran y ordenan títulos por volumen de visualizaciones, nivel de interacción u otros parámetros, aplicados a subconjuntos de usuarios (por país, por aplicación o por período de tiempo reciente). La necesidad de mantener esta información actualizada en todo momento es clave para ofrecer una experiencia personalizada y relevante a los usuarios, lo que hace crítico que las consultas subyacentes puedan ejecutarse de manera rápida y eficiente.

En el modelo relacional, la ejecución reiterada de este tipo de consultas puede verse afectada por la cantidad de datos involucrados y la complejidad de las relaciones entre tablas, especialmente en horas pico o con cargas de trabajo concurrentes. Además, el modelo relacional tiende a ser menos flexible frente a cambios frecuentes en la estructura de los datos o la aparición de nuevos atributos que deban incorporarse al análisis. Estas limitaciones motivaron la exploración de un modelo documental que, a través de técnicas como la desnormalización controlada y el uso de documentos embebidos, puede mejorar los tiempos de respuesta, reducir la necesidad de operaciones complejas y adaptarse mejor a los requerimientos dinámicos de este tipo de aplicaciones.

III. Modelado Documental

El diseño documental implementado en MongoDB se estructuró alrededor de una colección principal denominada `video_views`, que representa eventos individuales de visualización. Esta colección corresponde conceptualmente a la tabla de hechos del modelo relacional original, e incluye en cada documento los datos transaccionales relevantes: usuario, sesión, contenido, fecha, y las entidades asociadas (cliente, plataforma, dispositivo, etc.).

A diferencia del modelo relacional, en el esquema documental se optó por embeber los datos provenientes de dimensiones que presentan baja cardinalidad y poca variación a lo largo del tiempo, como client, device, location, partner, platform, y media. También se incluyó como subdocumento el campo date, que agrupa el código de fecha y su formato legible. Esta decisión fue clave para optimizar las consultas frecuentes, ya que permite ejecutar agregaciones directamente sobre una sola colección, sin necesidad de realizar joins costosos mediante `$lookup`.

Complementariamente, se definieron colecciones adicionales para representar entidades que requerían vistas consolidadas o independientes: `users`, `clients` y `media`. En el caso de `users`, por ejemplo, se eligió un modelo separado para permitir el análisis por país o tipo de dispositivo, sin tener que recorrer cientos o miles de visualizaciones por usuario. La colección `clients` se estructuró con un arreglo de partners embebidos para facilitar el análisis conjunto de clientes y sus plataformas de distribución. Finalmente, `media` se mantuvo como una colección independiente con los metadatos de cada título, permitiendo consultas directas sobre el catálogo, actualizaciones sin redundancia y una mejor integración con posibles interfaces de administración o dashboards.

III-A. Esquema de base de datos documental

Cada decisión de modelado se basó en los principios de desnormalización controlada y en la priorización del rendimiento en consultas frecuentes. Se procuró embeber datos que no cambian con frecuencia y cuya inclusión directa en el documento principal mejora significativamente la performance, como sucede con atributos del cliente, la plataforma o la categoría del contenido.

La colección `video_views` se convirtió así en el núcleo de análisis, permitiendo consultar directamente métricas clave como total de segundos visualizados por usuario, por país, por cliente o por tipo de contenido, sin necesidad de recorrer múltiples colecciones. Esta estructura facilita operaciones como `$group`, `$match`, `$sort` y `$unwind`, que son habituales en las consultas que sustentan funcionalidades críticas de la aplicación, como la generación de rankings o colecciones automáticas de títulos populares.

La separación de `users`, `clients` y `media` responde a necesidades de análisis más estratégicas: segmentación, actualización de catálogo, y exploración de cohortes de usuarios. Además, permite evitar la duplicación excesiva de información cuando se realizan tareas de perfilado o personalización. La estructura de las colecciones del modelo documental es la siguiente:

Listado 1 Esquema de la colección `video_views`

```
{
  "user_id": "string",
  "session_id": "string",
  "date": {
    "date_code": "integer",
    "full_date": "string"
  },
  "seconds_elapsed": "integer",
  "client": {
    "client_id": "string",
    "client_name": "string"
  },
  "device": {
    "device_type_id": "string",
    "device_type_name": "string"
  },
  "location": {
    "country_code": "string",
```

```
    "country_name": "string"
   },
   "partner": {
    "partner_id": "string",
    "partner_name": "string"
   },
   "platform": {
    "platform_id": "string",
    "platform_name": "string"
   },
   "media": {
    "media_id": "integer",
    "media_name": "string",
    "category_name": "string",
    "video_type": "string"
   }
}
```

Listado 2 Esquema de la colección `users`

```
{
  "user_id": "string",
  "countries": ["string"],
  "devices": ["string"]
}
```

Listado 3 Esquema de la colección `clients`

```
{
  "client_id": "string",
  "client_name": "string",
  "partners": [
    {
      "partner_id": "string",
      "partner_name": "string"
    }
  ]
}
```

Listado 4 Esquema de la colección `media`

```
{
  "media_id": "integer",
  "media_name": "string",
  "category_name": "string",
  "video_type": "string"
}
```

El modelo documental se diseñó con una fuerte orientación a la flexibilidad estructural, aprovechando la capacidad de MongoDB para representar relaciones jerárquicas mediante subdocumentos y arrays embebidos. Esto permite representar con naturalidad estructuras como date, client o partner dentro de cada visualización, facilitando el acceso rápido a los datos y reduciendo la necesidad de transformaciones adicionales.

Las entidades como `partner` o `platform`, que presentan un número acotado de valores por cliente, se embebieron como arreglos dentro de la colección `clients`. Esta estructura resulta especialmente útil para analizar el rendimiento por canal de distribución, e incluso para desarrollar reportes de revenue compartido entre clientes y plataformas, que es un caso de uso relevante en estas aplicaciones.

Además, al trabajar con colecciones separadas como `users`, se abre la posibilidad de incorporar métricas agregadas, como historial de países o tipos de dispositivo utilizados por un mismo usuario. Esta colección es clave para futuras tareas de clustering o sistemas de recomendación, que requieren una vista consolidada a nivel de usuario, pero no necesitan acceder a cada evento individual de visualización. Todas estas decisiones fueron guiadas por el análisis previo de las consultas más utilizadas en el entorno de producción: rankings de contenidos, vistas por país o cliente, y visualizaciones recientes. El nuevo esquema busca maximizar el rendimiento de estas consultas críticas, facilitar el mantenimiento de datos clave como catálogos de contenido y permitir una evolución fluida del modelo frente a futuros requerimientos de análisis o personalización.

Asimismo, durante el diseño del nuevo modelo se incorporaron algunos patrones clásicos de modelado en MongoDB. En la colección `users`, se aplicó el Subset Pattern, manteniendo una vista consolidada de cada usuario con el historial de países y dispositivos utilizados para acceder a las aplicaciones. Esta información es clave para análisis de perfilado y permite responder rápidamente a preguntas como desde qué países se conecta un usuario o cuántos tipos de dispositivos utiliza. En la colección `clients`, se utilizó una variante del Extended Reference Pattern, embebiendo directamente la información de los partners asociados (por ejemplo, FireTV, Android TV, marcas de televisores inteligentes como Samsung o LG). Esto permite identificar rápidamente los canales de distribución de cada aplicación, una métrica crítica para evaluar su alcance y optimizar decisiones de negocio relacionadas con acuerdos comerciales o disponibilidad técnica.

IV. Transformaciones realizadas

El proceso de transformación y construcción del nuevo modelo documental se realizó íntegramente en Python, utilizando `pymongo` para conectarse a una instancia de MongoDB Atlas donde ya se encontraba cargada la base de datos original en formato JSON. A diferencia de una transformación tradicional basada en archivos CSV, en este caso se partió directamente de documentos ya estructurados por tabla (uno por tabla del modelo relacional), los cuales fueron previamente subidos y almacenados como colecciones en MongoDB.

Desde un entorno colaborativo (Google Colab), se estableció la conexión con MongoDB Atlas utilizando credenciales de acceso y URI personalizado. A partir de allí, se ejecutaron múltiples scripts de transformación que re organizaron y enriquecieron los documentos existentes para construir las nuevas colecciones del modelo documental.

El pipeline de transformación se dividió en varias etapas, cada una correspondiente a la generación de una colección objetivo dentro del nuevo modelo. A continuación se describen los pasos principales:

1. Carga de datos desde MongoDB Atlas: Se conectó a la base de datos mediante pymongo, accediendo a las colecciones `fact_vod_uu_details`, `dim_client`, `dim_media`, entre otras. Estas representaban el modelo relacional original, cargado previamente a partir de documentos JSON (uno por tabla).
2. Transformación de la colección clients: A partir de la colección `dim_client`, se creó una nueva colección `clients`, enriquecida con campos agregados a través de pipelines. Se utilizó `$group` y `$addToSet` para reunir la lista de países (countries) y la lista de partners (partners) asociados a cada `client_id`, en base a los datos de visualización reales contenidos en `fact_vod_uu_details`. Estos resultados fueron luego actualizados directamente en los documentos de clients mediante operaciones `update_one`.
3. Transformación de la colección `video_views`: Esta fue la colección principal del nuevo modelo. Se creó a partir de los documentos en `fact_vod_uu_details`, re organizando los campos existentes para construir una estructura anidada y más expresiva. Se embebieron subdocumentos para `client`, `device`, `location`, `partner`, `platform`, `media` y `date`, estructurados jerárquicamente. Para manejar grandes volúmenes de datos, los documentos se insertaron por lotes en la colección final.
4. Transformación de la colección `media`: A partir de la colección `dim_media`, se generó la colección `media`, manteniendo únicamente los campos relevantes: `media_id`, `media_name`, `category_name` y `video_type`. Esta colección se pensó como una fuente centralizada de metadatos de contenido, útil para análisis o navegación del catálogo.
5. Transformación de la colección `users`: Se aplicó un pipeline de agregación sobre la colección `fact_vod_uu_details` para consolidar, por cada `uuid_hashvalue` (user ID), la lista de países (`country_code`) y tipos de dispositivos (`device_type_name`) utilizados. Luego, se proyectó el resultado para construir la colección `users`, que representa una vista resumida y útil para análisis de perfilado o segmentación.
6. Creación de índices: Se crearon índices específicos en cada colección, para facilitar búsquedas eficientes sobre atributos clave en estructuras embebidas.

El resultado final del proceso fue una base de datos documental compuesta por cuatro colecciones clave:

- `video_views`: documentos anidados que representan eventos individuales de visualización, con toda la información relevante embebida.


```
{'_id': ObjectId('683dfda82d64dba076265475'),
 'client': {'client_id': 'tg', 'client_name': 'Toon Goggles'},
 'date': {'date_code': 20250509, 'full_date': '2025-05-09'},
 'device': {'device_type_id': 'tv', 'device_type_name': 'TV'},
 'location': {'country_code': 'US', 'country_name': 'United States'},
 'media': {'category_name': 'N/A',
           'media_id': 284074,
           'media_name': 'Suzy and The Pothole',
           'video_type': 'Episode'},
 'partner': {'partner_id': 'viziosmartcast', 'partner_name': 'Vizio Smartcast'},
 'platform': {'platform_id': 'html5', 'platform_name': 'HTML5'},
 'seconds_elapsed': 358,
 'user_id': '4c8f120a1d08d1d94ec21dabbc0b3549f26ab431'}
```


Figura 2: Coleccion `Video_views`: documento de ejemplo

- `users`: vista agregada por usuario, con su historial de países y dispositivos.

```
{'_id': ObjectId('683f337fba2c293516e0b38a'),
 'countries': ['CM'],
 'devices': ['TV'],
 'user_id': '437b0a3476be144dd7796736a27ade8b1125de56'}
{'_id': ObjectId('683f337fba2c293516e0b38b'),
 'countries': ['VE'],
 'devices': ['TV'],
 'user_id': '3a316754b4bad1bbc5e6402b4f87863812bee98b'}
{'_id': ObjectId('683f337fba2c293516e0b38c'),
 'countries': ['US'],
 'devices': ['TV'],
 'user_id': '8f2c09ec8166f85ab2860aa0fd19d65b3e41db45'}
```


Figura 3: Coleccion `users`: documento de ejemplo

- `clients`: representación enriquecida de las aplicaciones desarrolladas, incluyendo los canales de distribución (partners) y países de uso.


```
{'_id': ObjectId('683377c02bda2666c61415ae'),
 'client_id': 'tg',
 'client_name': 'Toon Goggles',
 'countries': ['Greece',
               'Guadeloupe',
               'Chad',
               'Cyprus',
               'Seychelles',
               'Iran',
               'Croatia',
               'Nauru',
               'Tonga',
               'Lebanon'],
 'id': 69,
 'inserted_at': '2022-09-25 18:39:01.340033',
 'ocd': 1,
 'partners': [{'partner_id': 'skyworth', 'partner_name': 'Skyworth'},
              {'partner_id': 'samsungtv',
               'partner_name': 'Samsung Televisions'},
              {'partner_id': 'opera', 'partner_name': 'Opera'},
              {'partner_id': 'kurio', 'partner_name': 'Kurio'},
              {'partner_id': 'android', 'partner_name': 'General Android'},
              {'partner_id': 'zeasn', 'partner_name': 'Zeasn'},
              {'partner_id': 'bholenath', 'partner_name': 'Unknown'},
              {'partner_id': 'dyanora', 'partner_name': 'Unknown'},
              {'partner_id': 'marcel', 'partner_name': 'Unknown'},
              {'partner_id': 'sansui', 'partner_name': 'Unknown'}],
 'updated_at': '2023-08-29 08:00:20'}
```


Figura 4: Coleccion `clients`: documento de ejemplo. Se muestran los primeros 10 países y los primeros 10 partners para mejor visualizacion del documento.

- `media`: catálogo simplificado de los contenidos disponibles, con metadatos asociados.


```
{'_id': ObjectId('683e00f52d64dba076299da8'),
 'category_name': 'Unknown',
 'media_id': -99,
 'media_name': 'Unknown',
 'video_type': 'Unknown'}
{'_id': ObjectId('683e00f52d64dba076299da9'),
 'category_name': 'Drama',
 'media_id': 464,
 'media_name': 'MSROUT1',
 'video_type': 'Feature'}
{'_id': ObjectId('683e00f52d64dba076299daa'),
 'category_name': 'Comedia',
 'media_id': 458,
 'media_name': 'MSGIRL1',
 'video_type': 'Feature'}
```


Figura 5: Coleccion `media`: documento de ejemplo

V. Consultas e Índices

Las consultas implementadas buscaron imitar el resultado de las consultas implementadas previamente en SQL. Estas consultas no cuentan con una traducción lineal, ya que la estructura de tablas no es la misma entre SQL y los documentos de MongoDB. El objetivo de las consultas realizadas en Mongo DB fue el de obtener los mismos resultados que las consultas

SQL existentes. Este proceso requirió una adaptación, más que una traducción lineal, ya que existen diferencias en el modelado de datos entre SQL y MongoDB. Debido a la limitaciones de tamaño de los archivos que se podían colocar en la plataforma de Mongo DB, los datos subidos ya fueron filtrados por fecha y tipo de video, por lo que los filtros dd.full_date BETWEEN '20250501' AND '20250510' AND dm.video_type != 'Linear' no serán necesarios. La estructura del código de las consultas de Mongo DB sigue la misma lógica: la consulta sobre los documentos se hará en una variable denominada pipeline, luego el agreggate de esa sentencia se almacenará en una variable results que se recorre con un for para mostrar el resultado de la consulta. A continuación se detallan las consultas SQL y su similar en Mongo DB, se coloca solo el contenido de la consulta en Mongo DB, el código completo se puede contemplar en el apéndice.

V-A. Consulta 1

El match cumplirá el rol que ejecuta la cláusula where en filtrar los resultados, luego con ese resultado filtrado, se agrupa por media name, para a posteriori hacer la suma de los segundos. La cláusula project nos dice que columnas vamos a mostrar en nuestro resultado, funciona de la misma manera que un select. Por último ordenamos de mayor a menor y limitamos a 10 resultados

Consulta resumida Mongo DB

```
$match: { "client.client_id": "tg" }

$group: { _id: "$media.media_name", total_seconds: { $sum: "$seconds_elapsed" } }

$project: { _id: 0, media_name: "$_id", total_seconds: 1 }

$sort: { total_seconds: -1 }

$limit: 10
```

```
Top 10 contenidos por tiempo visto:
My Little Pony Equestria Girls: Rainbow Rocks => 3542125 segundos
Fire-Breathing Magic Dinosaur => 876877 segundos
Merry-go-round => 867021 segundos
Diamond Dog Caper => 830416 segundos
Beat the Raccoon => 636846 segundos
Pirates of Love => 614098 segundos
Magic wand => 588624 segundos
End of Year Special - Bubu and the Little Owls => 571157 segundos
The Missing Hero Crystal => 553976 segundos
Holidays Forever Compilation => 551834 segundos
```

Figura 6: Resultados MongoDB

Original SQL

```
   Original SQL:
SELECT
   dm.video\_type,
   dm.media\_name,
   SUM(fvd.seconds\_elapsed) AS total\_seconds
FROM
   fact\_vod\_uu\_details fvd
JOIN dim\_media dm ON fvd.dim\_media\_id = dm.id
JOIN dim\_date dd ON fvd.dim\_date\_code = dd.date\_code
WHERE
   dd.full\_date BETWEEN '20250501' AND '20250510'
   AND dm.client\_id = 'tg'
   AND dm.video\_type != 'Linear'
GROUP BY
   dm.video\_type, dm.media\_name
ORDER BY
   total\_seconds DESC
LIMIT 10
```

	video_type	media_name	total_seconds
0	Feature	My Little Pony Equestria Girls: Rainbow Rocks	3542125
1	Episode	Fire-Breathing Magic Dinosaur	876877
2	Episode	Merry-go-round	867021
3	Feature	Diamond Dog Caper	830416
4	Episode	Beat the Raccoon	636846
5	Episode	Pirates of Love	614098
6	Episode	Magic wand	588624
7	Feature	End of Year Special - Bubu and the Little Owls	571157
8	Episode	The Missing Hero Crystal	553976
9	Episode	Holidays Forever Compilation	551834

Figura 7: Resultados SQL

V-B. Consulta 2

La consulta funciona de manera similar a la consulta número 1 donde el match cumplirá el rol que ejecuta la cláusula where en filtrar los resultados, luego con ese resultado filtrado, en este caso se agrupa por client name, para luego hacer la suma de los segundos. La cláusula project nos dice que columnas vamos a mostrar en nuestro resultado, y por último ordenamos de mayor a menor por la cantidad de segundos.

Consulta resumida Mongo DB

```
$match: client.client\_id = "tg"
$group: \_id = client.client\_name, total\_seconds = sum(seconds\_elapsed)
$project: \_id = 0, client\_name = \_id, total\_seconds = 1
$sort: total\_seconds = -1
$limit: 10
```

```
Top 10 clientes por tiempo visto:
Toon Goggles => 92363525 segundos
```

Figura 8: Resultados MongoDB

Original SQL

```
   SELECT
   dc.client\_name,
   SUM(fvd.seconds\_elapsed) AS total\_seconds
FROM
   prod.fact\_vod\_uu\_details fvd
JOIN prod.dim\_client AS dc ON fvd.dim\_client\_id = dc.id
JOIN prod.dim\_date AS dd ON fvd.dim\_date\_code = dd.date\_code
INNER JOIN prod.dim\_media AS m ON m.id=fvd.dim\_media\_id
WHERE
   dd.full\_date BETWEEN '20250501' and '20250510'
   AND dc.client\_id='tg'
   AND m.video\_type!='Linear'
GROUP BY
   dc.client\_name
ORDER BY
   total\_seconds DESC
```

	client_name	total_seconds
0	Toon Goggles	92363525

Figura 9: Resultados SQL

V-C. Consulta 3

La consulta tomará los valores donde el cliente es Toon Google, el primer group nos permitirá agrupar por fecha y por user id, para identificar cada combinación única de usuario por día. Si un mismo usuario aparece varias veces en un mismo día, solo contará una vez. Posteriormente, se realiza una segunda agrupación. En esta etapa, el resultado de la agrupación anterior se vuelve a agrupar, pero esta vez solo por full_date. Por cada full_date, contamos cuántos pares únicos fecha, usuario resultaron de la primera agrupación. Esto nos da el número total de usuarios únicos para cada día. Luego project nos mostrará full date y el campo creado unique users, ordenamos por full date de menor a mayor.

Consulta resumida Mongo DB

```
$match: client.client\_id = "tg", video\_type  "Linear"
$group: \_id = { full\_date = date.full\_date, user\_id = user\_id }
$group: \_id = \_id.full\_date, unique\_users = sum(1)
$project: \_id = 0, full\_date = \_id, unique\_users = 1
$sort: full\_date = 1
```

```
2025-05-01 => 5729 usuarios únicos
2025-05-02 => 4530 usuarios únicos
2025-05-03 => 4866 usuarios únicos
2025-05-04 => 4546 usuarios únicos
2025-05-05 => 3957 usuarios únicos
2025-05-06 => 3914 usuarios únicos
2025-05-07 => 4009 usuarios únicos
2025-05-08 => 4092 usuarios únicos
2025-05-09 => 4189 usuarios únicos
2025-05-10 => 4843 usuarios únicos
```

Figura 10: Resultados MongoDB

Original SQL

```
   SELECT
   dd.full\_date,
   COUNT(DISTINCT uuid\_hashvalue) AS total\_users
FROM
   prod.fact\_vod\_uu\_details AS fvd

INNER JOIN prod.dim\_date dd ON fvd.dim\_date\_code = dd.date\_code
INNER JOIN prod.dim\_client AS c ON c.id=fvd.dim\_client\_id
INNER JOIN prod.dim\_media AS m ON m.id=fvd.dim\_media\_id

WHERE
dd.full\_date BETWEEN '20250501' AND '20250510'
AND c.client\_id='tg'
AND m.video\_type!='Linear'

GROUP BY
   dd.full\_date
ORDER BY
   dd.full\_date ASC;
```

	full_date	total_users
0	2025-05-01	5729
1	2025-05-02	4530
2	2025-05-03	4866
3	2025-05-04	4546
4	2025-05-05	3957
5	2025-05-06	3914
6	2025-05-07	4009
7	2025-05-08	4092
8	2025-05-09	4189
9	2025-05-10	4843

Figura 11: Resultados SQL

V-D. Consulta 4

Primero hacemos una dupla cliente partner, para cada combinación única, sumamos los segundos, luego mostramos los tres elementos ordenando por segundos totales. Al limitarlo a 10, generamos un top 10 de combinaciones cliente–partner con más segundos vistos. No necesariamente son los 10 clientes o las 10 plataformas más vistas por separado, sino las combinaciones más vistas.

Consulta resumida Mongo DB

```
$group: \_id = { client\_name = client\_name, partner\_name = partner\_name }, total\_seconds = sum(seconds\_watched)
$project: \_id = 0, client\_name = \_id.client\_name, partner\_name = \_id.partner\_name, total\_seconds = 1
$sort: total\_seconds = -1
$limit: 10
```

```
Toon Goggles => Hisense, 27459163
Toon Goggles => Roku, 13100312
Toon Goggles => Vizio Smartcast, 9557878
Toon Goggles => TCL, 9469610
Toon Goggles => M Star, 7468756
Toon Goggles => General Android, 4024805
Toon Goggles => Philips, 3668155
Toon Goggles => Unknown, 2898904
Toon Goggles => General HTML5, 1845031
Toon Goggles => Hisense Legacy, 1665119
```

Figura 12: Resultados MongoDB

Original SQL

```
    SELECT
    dc.client\_name,
    dp.partner\_name,
    SUM(fvd.seconds\_elapsed) AS total\_seconds
FROM
    prod.fact\_vod\_uu\_details AS fvd

INNER JOIN prod.dim\_date AS dd ON dd.date\_code=fvd.dim\_date\_code
INNER JOIN prod.dim\_client dc ON fvd.dim\_client\_id = dc.id
INNER JOIN prod.dim\_media AS m ON m.id=fvd.dim\_media\_id
INNER JOIN prod.dim\_partner dp ON fvd.dim\_partner\_id = dp.id

WHERE
dd.full\_date BETWEEN '20250501' AND '20250510'
AND dc.client\_id='tg'
AND m.video\_type!='Linear'
```

	client_name	partner_name	total_seconds
0	Toon Goggles	Hisense	27459163
1	Toon Goggles	Roku	13100312
2	Toon Goggles	Vizio Smartcast	9557878
3	Toon Goggles	TCL	9469610
4	Toon Goggles	M Star	7468756
5	Toon Goggles	General Android	4024805
6	Toon Goggles	Philips	3668155
7	Toon Goggles	Unknown	2898904
8	Toon Goggles	General HTML5	1845031
9	Toon Goggles	Hisense Legacy	1665119

Figura 13: Resultados SQL

VI. Índices creados y su justificación

Indice Client Name - Media Name

Este índice compuesto, que incluye client.client_name y media.media_name en ese orden, fue diseñado para optimizar el rendimiento de consultas que cruzan datos entre clientes y contenidos multimedia. La lógica detrás de este orden es que

MongoDB construye una estructura de acceso que primero organiza los datos por cliente y, dentro de cada cliente, los organiza por contenido. Esto lo hace muy eficiente para consultas que agrupan o filtran por cliente inicialmente y luego por contenido. Es útil para identificar los contenidos más vistos por cada cliente o para analizar el consumo de títulos específicos por segmento de usuario. En la práctica, este índice mejora la eficiencia de consultas como ”top títulos por cliente”. Por ejemplo, una operación de agregación que agrupa por cliente y título para sumar los segundos visualizados (seconds_elapsed). Si se aplica un filtro inicial por un cliente específico (ej., ”BDNR”) antes de agrupar por los títulos vistos por ese cliente, MongoDB puede utilizar el índice de forma completa, evitando un escaneo de toda la colección y mejorando significativamente los tiempos de respuesta, especialmente con grandes volúmenes de datos. Este ejemplo necesitaría un código similar a este:

```
pipeline = [
   { "$match": { "client.client\_name": "BDNR" } },
   { "$group": {
      "\_id": "$media.media\_name",
      "total\_seconds": { "$sum": "$seconds\_elapsed" }
   }},
   { "$sort": { "total\_seconds": -1 } }
]
```

Es importante destacar que este índice no aporta beneficios a consultas que solo utilizan media.media_name sin considerar el cliente. Si el campo client.client_name es el primer componente del índice, MongoDB no puede ”saltar.^este campo para buscar únicamente por media.media_name. Para esos casos, sería necesario crear un índice específico sobre media.media_name si se busca mejorar el rendimiento.

Codigo de creación del indice:

```
db.video\_views.create\_index([
   ("client.client\_name", 1),
   ("media.media\_name", 1)
])
```

Índice Media Name - Video Type

La creación de un índice compuesto sobre media.media_name y media.video_type está orientada a optimizar consultas analíticas que busquen entender el comportamiento de consumo por tipo de contenido dentro del catálogo de cada creador. Este índice resulta útil en escenarios donde un mismo contenido está disponible en múltiples formatos, como episodios, cortos, películas. Al estructurarlo como una tupla, el índice permite diferenciar el rendimiento de cada tipo de video asociado a un título específico según el formato.Por ejemplo, es frecuente que programás largos que se enfocan en conversaciones tengan un mejor rendimiento en shorts que en toda una entrevista o dialogo. Esto habilita estudios más precisos sobre preferencias de los usuarios, efectividad de ciertos tipos de contenido y decisiones editoriales como por ejemplo priorizar la producción de cortos frente a episodios largos. Una lógica similar aplica al índice compuesto por client.client_name y platform.platform_name, donde el cliente puede analizar como consumen sus clientes y a partir de ellos, como mostrar su contenido, por ejemplo si su consumo mayoritario es de shorts, la grabacion de los videos deberá estar pensada para ser mostrada en formato vertical. Combinando ambos indices podemos llegar a evaluar si el tipo de video y plataforma estan relacionados, podemos suponer que los usuarios veran shorts en sus celulares, y que episodios se veran en computadoras o televisiones, pero que peliculas tendran un mayor presencia en un consumo por TV,

```
db.video\_views.create\_index([
   ("media.media\_name", 1),
   ("media.video\_type", 1)
])

Situacin similar aplica para los indices:

db.video\_views.create\_index([
   ("client.client\_name", 1),
   ("platform.platform\_name", 1)
])
```

VII. Análisis de Performance

Comparación de tiempos de ejecución entre los dos modelos

Tiempo en mms

Consulta	SQL Redshift	SQL Colab	MongoDB
1	4779	537	677
2	9742	230.2	357
3	13756	431.1	519
4	6263	76603	448

Para evaluar el rendimiento comparativo entre el modelo relacional y el modelo documental, inicialmente ejecutamos las consultas SQL sobre la base de datos relacional en su entorno de producción original: un clúster de Redshift en AWS. Este entorno cuenta con todos los recursos necesarios (motor, servidor y configuración específica) para operar a escala con grandes volúmenes de datos, y es donde actualmente reside el modelo relacional activo.

Sin embargo, para obtener una comparación más equilibrada con respecto a las consultas realizadas sobre MongoDB Atlas desde Google Colab (utilizando Python y la API de pymongo), decidimos construir un segundo entorno más controlado y homogéneo. En este caso, desarrollamos un script adicional en Google Colab que replica el modelo relacional utilizando los mismos archivos JSON previamente cargados en MongoDB. Estos archivos fueron importados como DataFrames de pandas, y luego transformados a tablas utilizando sqlite3, lo que permitió ejecutar las consultas SQL originales directamente dentro del entorno de Colab.

De esta forma, trabajamos con tres escenarios diferenciados:

1. El entorno de Redshift (producción).
2. El entorno simulado relacional en SQLite desde Colab.
3. El entorno documental en MongoDB Atlas accedido desde Colab con Python.

A partir de estos tres escenarios, se ejecutaron las mismas consultas representativas y se registraron los tiempos de ejecución en cada caso. Esto permitió una comparación más justa del rendimiento entre los modelos, considerando tanto el contexto real de uso como un entorno controlado común.

Al comparar el rendimiento, las consultas SQL en Colab mostraron una mejor performance para tres de las cuatro consultas. No obstante, la diferencia en tiempo se midió en milisegundos, lo que indica un impacto marginal. La distinción significativa en favor del modelado en MongoDB se observa en la cuarta consulta, donde el tiempo de ejecución se redujo a aproximadamente un 10 por ciento del tiempo total requerido para la misma consulta en SQL.

Esto nos permite remarcar la eficiencia que MongoDB puede ofrecer en escenarios específicos de consulta. La consulta original contaba con varios INNER JOIN mientras que en Mongo esto se logra consultando una única colección.

Este resultado evidencia uno de los principales beneficios del modelo documental: al tener todos los datos anidados dentro del mismo documento (client, partner, media, etc.), MongoDB puede ejecutar el pipeline de agregación de forma directa y en memoria, sin joins ni reconstrucciones inter-tablas.

Este tipo de operación, en un esquema SQL, requiere múltiples joins entre fact_vod_uu_details, dim_client, dim_partner, dim_media y dim_date, lo cual introduce un sobrecosto importante, sobre todo si las relaciones no están optimizadas con índices compuestos adecuados. Otro beneficio observado tiene que ver con la naturaleza embebida del modelo de datos en MongoDB. Por ejemplo, los campos client.client_name, media.media_name o partner.partner_name ya están presentes en cada documento.

En conclusión, aunque MongoDB no superó a SQL en la totalidad de las consultas, su ventaja se hizo evidente cuando se involucran estructuras de datos complejas y relaciones diversas, tal como se observó en la consulta 4. El modelo documental de MongoDB reduce la necesidad de operaciones de ”join”, facilita el acceso a datos anidados y aprovecha de forma más eficiente el flujo de las agregaciones.

VIII. Conclusiones

Las cuatro consultas ejecutadas son parte de la operativa diaria, por lo que las mejoras de rendimiento se plasmarán en un caso de uso real, donde las queries sencillas en una misma tabla no tienen una mejora que amerite el esfuerzo de realizar

una migración pero a medida que la complejidad de las queries crece, y el volumen de datos también, la ventaja de tener los datos estructurados como documentos empieza a tener mayor claridad y en caso de que este tipo de análisis se hagan más frecuentemente, el esfuerzo de realizar una migración de esquemas puede cobrar sentido para el negocio.

Resumen de hallazgos y beneficios del modelo documental

La transición de un modelo relacional tradicional a un esquema documental en MongoDB ha evidenciado mejoras tangibles en rendimiento, simplicidad operativa y adecuación a las demandas analíticas actuales. Un punto clave fue la marcada optimización en la consulta 4, donde el tiempo de ejecución se redujo significativamente, pasando de más de 76.000 milisegundos en SQL a tan solo 448 milisegundos en MongoDB.

Esta mejora se fundamenta en la naturaleza del modelo documental. Al consolidar datos como client, partner y media dentro de un único documento, se eliminan los costos inherentes a los múltiples joins que son necesarios en SQL. Esta característica permitió a MongoDB procesar las agregaciones directamente en memoria sobre una única colección, lo que optimiza tanto el tiempo de procesamiento como el uso de recursos.

Adicionalmente, se constató que el nuevo esquema facilita la lectura y la construcción de consultas, ya que los datos se mantienen agrupados contextualmente. Consultas como el ranking de contenidos por cliente o la identificación de usuarios únicos por día resultaron más directas de implementar y escalar en MongoDB.

Lecciones aprendidas y posibles mejoras futuras

Se evidenció que la ventaja de MongoDB no se materializa necesariamente en consultas simples o de bajo volumen, donde SQL mantiene su solidez. La mejora en MongoDB se vuelve tangible cuando se trabaja con estructuras complejas, relaciones múltiples o necesidades de respuesta en tiempo real. A futuro, se podría implementar índices compuestos adicionales cuando se expanda el dataset completo (por ejemplo, combinaciones de cliente, video_type y fecha), lo cual permitirá escalar el modelo sin penalizar la performance.Otra mejora posible consiste en enriquecer la colección users para soportar tareas de segmentación y clustering, e integrar pipelines automatizados de actualización de datos. Finalmente, se destaca como aspecto a continuar desarrollando la incorporación de monitoreo y profiling sistemático de consultas para guiar decisiones de modelado e indexación a medida que crezca el volumen de datos y se diversifiquen los reportes de negocio.

IX. Apéndice

Repositorio github

En este repositorio se puede encontrar los notebooks y las bases de datos necesarias para reproducir el trabajo presentado en este informe.

https://github.com/joanauriello/BDNR

Código completo Mongo DB

Consulta 1

```
pipeline = [
   {
      "$match": {
         "client.client\_id": "tg"
      }
   },
   {
      "$group": {
         "\_id": "$media.media\_name",
         "total\_seconds": { "$sum": "$seconds\_elapsed" }
      }
   },
   {
      "$project": {
         "\_id": 0,
         "media\_name": "$\_id",
         "total\_seconds": 1
      }
   },
   {
      "$sort": {
         "total\_seconds": -1
      }
   },
   {
      "$limit": 10
   }
]

# Ejecutamos el pipeline
```

```
results = db.video\_views.aggregate(pipeline)

# Paso 2: Mostramos los resultados
print("Top 10 contenidos por tiempo visto:")
for doc in results:
   media\_name = doc["media\_name"]
   total\_seconds = doc["total\_seconds"]
   print(f"{media\_name} => {total\_seconds} segundos")
```

Consulta 2

```
pipeline = [
   {
      "$match": {
         "client.client\_id": "tg"
      }
   },
   {
      "$group": {
         "\_id": "$client.client\_name",
         "total\_seconds": { "$sum": "$seconds\_elapsed" }
      }
   },
   {
      "$project": {
         "\_id": 0,
         "client\_name": "$\_id",
         "total\_seconds": 1
      }
   },
   {
      "$sort": {
         "total\_seconds": -1
      }
   },
   {
      "$limit": 10
   }
]

# Ejecutamos el pipeline
results = db.video\_views.aggregate(pipeline)

# Paso 2: Mostramos los resultados
print("Top 10 clientes por tiempo visto:")
for doc in results:
   client\_name = doc["client\_name"]
   total\_seconds = doc["total\_seconds"]
   print(f"{client\_name} => {total\_seconds} segundos")
```

Consulta 3

```
pipeline = [
   # 1. Filtramos los documentos que interesan
   {
      "$match": {
         "client.client\_id": "tg", # campo anidado dentro de client
         "video\_type": { "$ne": "Linear" } # mismo criterio que usabas antes
      }
   },

   # 2. Identificamos cada usuario distinto por da
   {
      "$group": {
         "\_id": {
            "full\_date": "$date.full\_date", # fecha (viene dentro del sub-documento date)
            "user\_id": "$user\_id" # identificador nico del usuario
         }
      }
   },

   # 3. Contamos cuntos usuarios nicos hubo en cada da
   {
      "$group": {
         "\_id": "$\_id.full\_date",
         "unique\_users": { "$sum": 1 }
      }
   },

   # 4. Formateamos la salida
```

```
    {
        "$project": {
            "\_id": 0,
            "full\_date": "$\_id",
            "unique\_users": 1
        }
    },

    # 5. Orden cronolgico ascendente
    { "$sort": { "full\_date": 1 } }
]

results = db.video\_views.aggregate(pipeline)

for doc in results:
    print(f"{doc['full\_date']} => {doc['unique\_users']} usuarios nicos")
```

Consulta 4

```
pipeline = [
    {
        "$group": {
            "\_id": {
                "client\_name": "$client\_name",
                "partner\_name": "$partner\_name"
            },
            "total\_seconds": { "$sum": "$seconds\_watched" }
        }
    },
    {
        "$project": {
            "\_id": 0,
            "client\_name": "$\_id.client\_name",
            "partner\_name": "$\_id.partner\_name",
            "total\_seconds": 1
        }
    },
    {
        "$sort": {
            "total\_seconds": -1
        }
    },
    {
        "$limit": 10
    }
]


results = db.fact\_vod\_uu\_details.aggregate(pipeline)

for doc in results:
    print(f"{doc['client\_name']} => {doc['partner\_name']}, {doc['total\_seconds']}")
```