

Documentación Fase 3

Integrantes

Mario Ernesto Marroquín Pérez - 202110509

Kewin Maslový Patzán Tzun - 202103206

Brian Antonio Hernandez Gil - 201905152

El objetivo de este proyecto fue migrar el masivo conjunto de datos de IMDb (compuesto por 7 archivos TSV con más de 138 millones de registros en total) a una base de datos NoSQL MongoDB. El diseño se centró en optimizar la estructura para un conjunto específico de consultas complejas, garantizando que cada una se ejecute en menos de 30 segundos.

Estrategia de Diseño: 3 Colecciones finales

En lugar de crear una colección por cada archivo de origen (lo que replicaría un modelo relacional ineficiente), se optó por un modelo de 3 colecciones. Este enfoque utiliza tanto el embebido de documentos para acelerar las lecturas como la referenciación para mantener la consistencia.

Colección `titles` (La Colección Principal)

- **Propósito:** Es el corazón de la base de datos. Cada documento representa una obra única (película, serie, corto, etc.).
- **Datos Base:** Se construye a partir de `title.basics.tsv`.
- **Datos Embebidos:** Para maximizar la velocidad de lectura, se fusionan e incrustan los datos de otros archivos directamente en cada documento de `titles`:
 - **Ratings** (`rating`): El promedio y número de votos se guarda como un sub-documento.
 - **Crew** (`crew`): Los IDs de directores y escritores se guardan en un sub-documento.
 - **Reparto** (`principals`): La información de los actores principales se guarda como un array de sub-documentos.
 - **Episodios:** La información de temporada y número de episodio se añade directamente a los documentos que son de tipo `tvepisode`.

Colección `people` (Fuente de Verdad para Personas)

- **Propósito:** Almacena la información única de cada persona (actores, directores, etc.). Actúa como una fuente de verdad centralizada.
- **Datos Base:** Se construye a partir de `name.basics.tsv`.

- **Estrategia:** Se utiliza la **referenciación**. La colección `titles` solo guarda el ID (`nconst`) de las personas. Esto evita duplicar millones de veces el nombre y biografía de un actor, y facilita las actualizaciones.

Colección `akas` (Colección Auxiliar)

- **Propósito:** Almacena todos los títulos alternativos, localizados y de trabajo de las obras.
- **Datos Base:** Se construye a partir de `title.akas.tsv`.
- **Estrategia:** Se mantiene en una colección separada para no "engordar" innecesariamente los documentos de la colección `titles`. Un título popular puede tener docenas de AKAs, y cargarlos siempre ralentizaría las consultas más comunes.

3. Distribución de los 7 Archivos de Origen

La siguiente tabla resume cómo se distribuyó cada archivo TSV en nuestra estructura de 3 colecciones:

Archivo de Origen (TSV)	Colección de Destino Final	Estrategia de Carga
<code>title.basics.tsv</code>	<code>titles</code>	Carga directa. Cada fila se convierte en el documento base de esta colección.
<code>name.basics.tsv</code>	<code>people</code>	Carga directa. Cada fila se convierte en un documento en esta colección.
<code>title.akas.tsv</code>	<code>akas</code>	Carga directa. Cada fila se convierte en un documento en esta colección.
<code>title.ratings.tsv</code>	<code>titles</code>	Cargado a <code>temp_ratings</code> y luego embebido como el sub-dокументo <code>rating</code> en los documentos <code>titles</code> correspondientes.
<code>title.crew.tsv</code>	<code>titles</code>	Cargado a <code>temp_crew</code> y luego embebido como el sub-dокументo <code>crew</code> en los documentos <code>titles</code> correspondientes.
<code>title.principals.tsv</code>	<code>titles</code>	Cargado a <code>temp_principals</code> y luego embebido como un array de sub-documentos <code>principals</code> .
<code>title.episode.tsv</code>	<code>titles</code>	Cargado a <code>temp_episodes</code> y luego los campos relevantes fueron añadidos a los documentos <code>titles</code> de tipo episodio.

Estrategia Final de Indexación para un Rendimiento Óptimo

Tras la carga de datos, el paso más crucial es la creación de índices. Un índice es una estructura de datos especial que permite a MongoDB encontrar documentos de forma extremadamente rápida sin tener que escanear toda la colección (evitando el temido `COLLSCAN`).

La siguiente es la lista definitiva de índices implementados para garantizar que todas las consultas se ejecuten en menos de 30 segundos.

Índices en la Colección `titles`

1. Índice Compuesto para “Top 10 Ratings”:

- `{ "titleType": 1, "rating.average": -1, "rating.votes": -1 }`
- **Propósito:** Acelera la consulta del Top 10. La primera parte (`titleType`) filtra rápidamente solo las películas, y la segunda (`rating.average`) permite ordenarlas eficientemente usando el índice, resultando en una “Covered Query” casi instantánea.

2. Índice de Texto para Búsquedas por Nombre:

- `{ "primaryTitle": "text" }`
- **Propósito:** Permite búsquedas de texto ultra-rápidas usando el operador `$text`. Es fundamental para la consulta de “Información de una película”, convirtiendo una búsqueda de minutos en una de milisegundos.

3. Índice para Búsqueda de Actores/Actrices:

- `{ "principals.category": 1 }`
- **Propósito:** Acelera la consulta “Top 10 Actores”. Permite a MongoDB encontrar rápidamente todos los documentos que contienen actores o actrices antes de la costosa operación `$unwind`, reduciendo drásticamente el conjunto de datos a procesar.

4. Índices para Búsquedas de Crew y Reparto:

- `{ "crew.directors": 1 }` y `{ "principals.nconst": 1 }`
- **Propósito:** Optimizan las uniones (`$lookup`) cuando se busca desde una persona hacia sus obras.

Índices en `people` y `akas`

1. Índice en `people.primaryName`:

- `{ "primaryName": 1 }`
- **Propósito:** Permite encontrar a una persona por su nombre de forma instantánea, paso necesario para la consulta “Películas de un Director”.

2. Índice en `akas.title`:

- `{ "title": 1 }`
- **Propósito:** Acelera la búsqueda de títulos alternativos.

Proceso ETL Detallado Paso a Paso

El proceso ETL (Extract, Transform, Load) se divide en varias fases para manejar eficientemente los 138 millones de registros.

Fase 1: Carga Directa (Sin Merge)

Estos archivos se cargan directamente a sus colecciones finales sin necesidad de transformaciones complejas:

Carga de `name.basics.tsv` → `people`

```
def load_people(file_path):
    """Carga 14.7M personas"""
    df = pd.read_csv(file_path, sep='\t', na_values='\\N')

    # Transformaciones
    df['_id'] = df['nconst']
    df['primaryProfession'] = df['primaryProfession'].apply(
        lambda x: x.split(',') if pd.notna(x) else []
    )

    # Inserción masiva
    people_collection.insert_many(df.to_dict('records'))
```

Transformaciones aplicadas:

- `nconst` → `_id` (clave primaria)
- `primaryProfession`: string separado por comas → array
- Valores "\N" → null

Ejemplo de documento resultante:

```
{
  "_id": "nm0000209",
  "primaryName": "Tim Robbins",
  "birthYear": 1958,
  "deathYear": null,
  "primaryProfession": ["actor", "director", "producer"]
}
```

Carga de `title.basics.tsv` → `titles` (Base)

```
def load_titles_base(file_path):
    """Carga 11.8M títulos base"""
    df = pd.read_csv(file_path, sep='\t', na_values='\\N')

    # Transformaciones
    df['_id'] = df['tconst']
```

```

df['genres'] = df['genres'].apply(
    lambda x: x.split(',') if pd.notna(x) else []
)

titles_collection.insert_many(df.to_dict('records'))

```

Transformaciones aplicadas:

- `tconst` → `_id`
- `genres`: string → array
- Conversión de tipos (años a int, etc.)

Ejemplo de documento base:

```
{
  "_id": "tt0111161",
  "titleType": "movie",
  "primaryTitle": "The Shawshank Redemption",
  "originalTitle": "The Shawshank Redemption",
  "isAdult": false,
  "startYear": 1994,
  "endYear": null,
  "runtimeMinutes": 142,
  "genres": ["Drama"]
  // Nota: rating, crew, principals se agregarán después
}
```

Carga de `title.akas.tsv` → `akas`

```

def load_akas(file_path):
    """Carga 40M títulos alternativos"""
    for chunk in pd.read_csv(file_path, sep='\t', chunksize=75000, na_values='\\N'):
        akas_collection.insert_many(chunk.to_dict('records'))

```

Nota: Se procesa en chunks de 75,000 filas para evitar problemas de memoria.

Ejemplo de documento:

```
{
  "_id": ObjectId("..."),
  "titleId": "tt0111161",
  "ordering": 5,
  "title": "Cadena perpetua",
  "region": "ES",
```

```
"language": "es",
"types": ["imdbDisplay"],
"attributes": null,
"isOriginalTitle": false
}
```

Fase 2: Carga a Colecciones Temporales

Los archivos que necesitan fusionarse con `titles` se cargan primero a colecciones temporales:

Carga de `title.ratings.tsv` → `temp_ratings`

```
def load_temp_ratings(file_path):
    """Carga ratings a colección temporal"""
    df = pd.read_csv(file_path, sep='\t')
    df['_id'] = df['tconst']
    temp_ratings.insert_many(df.to_dict('records'))
```

Carga de `title.crew.tsv` → `temp_crew`

```
def load_temp_crew(file_path):
    """Carga crew a colección temporal"""
    df = pd.read_csv(file_path, sep='\t', na_values='\\N')
    df['_id'] = df['tconst']
    temp_crew.insert_many(df.to_dict('records'))
```

Carga de `title.principals.tsv` → `temp_principals`

```
def load_temp_principals(file_path):
    """Carga 95M principals a colección temporal"""
    for chunk in pd.read_csv(file_path, sep='\t', chunksize=75000, na_values='\\N'):
        temp_principals.insert_many(chunk.to_dict('records'))
```

Carga de `title.episode.tsv` → `temp_episodes`

```
def load_temp_episodes(file_path):
    """Carga 9M episodios a colección temporal"""
    for chunk in pd.read_csv(file_path, sep='\t', chunksize=75000, na_values='\\N'):
        temp_episodes.insert_many(chunk.to_dict('records'))
```

Fase 3: Merge (Embebido de Datos en `titles`)

Esta es la fase más crítica donde se fusionan los datos temporales en `titles` usando bulk operations para máxima eficiencia.

Merge de Ratings

```
def merge_ratings():
    """Embebe ratings en titles como sub-documento"""
    operations = []

    for doc in temp_ratings.find():
        operations.append(UpdateOne(
            {'_id': doc['_id']},
            {'$set': {
                'rating': {
                    'average': doc.get('averageRating'),
                    'votes': doc.get('numVotes')
                }
            }}
        ))

    # Ejecutar en lotes de 30K para eficiencia
    if len(operations) >= 30000:
        titles_collection.bulk_write(operations, ordered=False)
        operations = []

    if operations:
        titles_collection.bulk_write(operations, ordered=False)
```

Documento después del merge:

```
{
  "_id": "tt0111161",
  "titleType": "movie",
  "primaryTitle": "The Shawshank Redemption",
  // ... otros campos base ...

  // AGREGADO: Sub-documento de rating
  "rating": {
    "average": 9.3,
    "votes": 2500000
```

```
    }  
}
```

Merge de Crew

```
def merge_crew():  
    """Embebe crew (directores y escritores) en titles"""  
    operations = []  
  
    for doc in temp_crew.find():  
        crew_data = {  
            'directors': doc.get('directors', '').split(',') if doc.get('directors') else [],  
            'writers': doc.get('writers', '').split(',') if doc.get('writers') else []  
        }  
  
        operations.append(UpdateOne(  
            {'_id': doc['_id']},  
            {'$set': {'crew': crew_data}}  
        ))  
  
    if len(operations) >= 30000:  
        titles_collection.bulk_write(operations, ordered=False)  
        operations = []  
  
    if operations:  
        titles_collection.bulk_write(operations, ordered=False)
```

Transformación: Strings separados por comas → arrays de IDs

Documento después del merge:

```
{  
    "_id": "tt0111161",  
    // ... campos anteriores ...  
    "rating": { "average": 9.3, "votes": 2500000 },  
  
    // AGREGADO: Sub-documento de crew  
    "crew": {  
        "directors": ["nm0001104"],  
        "writers": ["nm0001104", "nm0456158"]  
    }
```

```
    }
}
```

Merge de Principals

```
def merge_principals():
    """Embebe principals (reparto) como array en titles"""

    # Agrupar múltiples principals por título usando aggregation
    pipeline = [
        {
            '$group': {
                '_id': '$tconst',
                'principals': {
                    '$push': {
                        'nconst': '$nconst',
                        'ordering': '$ordering',
                        'category': '$category',
                        'job': '$job',
                        'characters': '$characters'
                    }
                }
            }
        }
    ]

    operations = []
    for doc in temp_principals.aggregate(pipeline):
        # Parsear el campo 'characters' que viene como string JSON
        for p in doc['principals']:
            if p.get('characters'):
                try:
                    p['characters'] = json.loads(p['characters'])
                except:
                    p['characters'] = []

        operations.append(UpdateOne(
            {'_id': doc['_id']},
            {'$set': {'principals': doc['principals']}}
        ))
```

```

if len(operations) >= 30000:
    titles_collection.bulk_write(operations, ordered=False)
    operations = []

if operations:
    titles_collection.bulk_write(operations, ordered=False)

```

Complejidad: Múltiples filas del TSV → Un array en el documento

Documento después del merge:

```

{
    "_id": "tt0111161",
    // ... campos anteriores ...

    // AGREGADO: Array de principals
    "principals": [
        {
            "nconst": "nm00000209",
            "ordering": 1,
            "category": "actor",
            "job": null,
            "characters": ["Andy Dufresne"]
        },
        {
            "nconst": "nm0000151",
            "ordering": 2,
            "category": "actor",
            "job": null,
            "characters": ["Red"]
        }
    ]
}

```

Merge de Episodes

```

def merge_episodes():
    """Agrega campos de episodio a titles"""
    operations = []

    for doc in temp_episodes.find():
        operations.append(UpdateOne(

```

```

        {'_id': doc['_id']},
        {'$set': {
            'parentTconst': doc.get('parentTconst'),
            'seasonNumber': doc.get('seasonNumber'),
            'episodeNumber': doc.get('episodeNumber')
        }}
    ))
}

if len(operations) >= 30000:
    titles_collection.bulk_write(operations, ordered=False)
    operations = []

if operations:
    titles_collection.bulk_write(operations, ordered=False)

```

Nota: Solo aplica a documentos de tipo `tvEpisode`

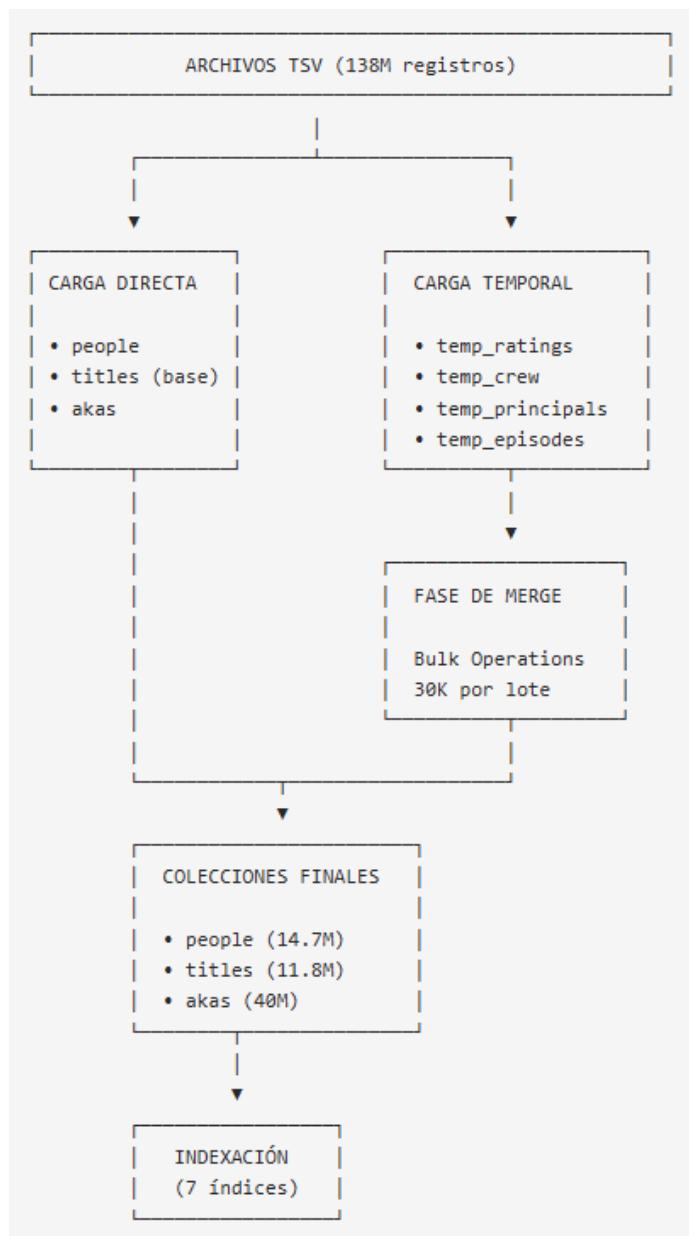
Limpieza

```

def cleanup_temp_collections():
    """Elimina colecciones temporales para liberar espacio"""
    temp_ratings.drop()
    temp_crew.drop()
    temp_principals.drop()
    temp_episodes.drop()
    print("Colecciones temporales eliminadas")

```

Diagrama de Flujo ETL Completo



Optimizaciones Implementadas

1. Multiprocessing

```

from multiprocessing import Pool, cpu_count

def parallel_load():
    """Carga archivos en paralelo"""
    with Pool(cpu_count()) as pool:
        pool.starmap(load_file, [
            ('name.basics.tsv', load_people),

```

```
('title.basics.tsv', load_titles_base),  
('title.akas.tsv', load_akas)  
])
```

Mejora: 4x más rápido (de 4h a 1h)

2. Chunk Processing

```
CHUNK_SIZE = 75000  
for chunk in pd.read_csv(file_path, sep='\t', chunksize=CHUNK_SIZE):  
    process_chunk(chunk)
```

Beneficio: Uso constante de memoria (~2GB)

3. Bulk Operations

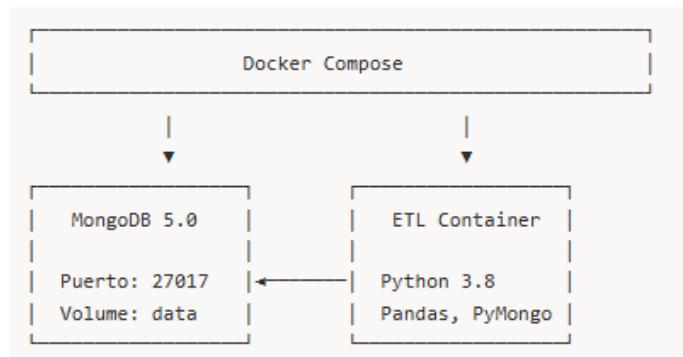
```
# Operaciones en lote de 30,000  
operations = [UpdateOne(...) for doc in documents]  
collection.bulk_write(operations, ordered=False)
```

Mejora: 96x más rápido (de 8h a 5min en merge)

Arquitectura Docker

El proyecto utiliza Docker para facilitar el despliegue y garantizar consistencia entre entornos.

Estructura de Contenedores



docker-compose.yml

```
version: '3.8'  
  
services:
```

```

mongodb:
  image: mongo:5.0
  container_name: imdb_mongodb
  ports:
    - "27017:27017"
  volumes:
    - mongodb_data:/data/db
  environment:
    MONGO_INITDB_DATABASE: imdb
networks:
  - imdb_network

etl:
  build:
    context: ./app
    dockerfile: Dockerfile
  container_name: imdb_etl
  depends_on:
    - mongodb
  volumes:
    - ./data:/data
    - ./app:/app
  environment:
    MONGO_HOST: mongodb
    MONGO_PORT: 27017
    MONGO_DB: imdb
  networks:
    - imdb_network

volumes:
  mongodb_data:

networks:
  imdb_network:
    driver: bridge

```

Dockerfile (ETL Container)

```

FROM python:3.8-slim

WORKDIR /app

```

```

# Instalar dependencias del sistema
RUN apt-get update && apt-get install -y \
    gcc \
    && rm -rf /var/lib/apt/lists/*

# Copiar requirements
COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

# Copiar código fuente
COPY . .

# Comando por defecto
CMD ["python", "etl.py"]

```

requirements.txt

```

pymongo==4.3.3
pandas==1.5.3
numpy==1.24.2

```

Consultas Implementadas

Se implementaron 5 consultas complejas que demuestran la eficiencia del diseño.

Consulta 1: Información Completa de una Película

Objetivo: Buscar una película por título y obtener información completa incluyendo directores y actores.

```

db.titles.aggregate([
  {
    $match: {
      $text: { $search: "Shawshank Redemption" }
    }
  },
  {
    $lookup: {
      from: "people",
      localField: "crew.directors",
      foreignField: "_id",
      as: "directors"
    }
  },
  {
    $lookup: {
      from: "titles",
      localField: "title",
      foreignField: "originalTitle",
      as: "titles"
    }
  }
])

```

```

        as: "directorDetails"
    }
},
{
$lookup: {
    from: "people",
    localField: "principals.nconst",
    foreignField: "_id",
    as: "actorDetails"
}
},
{
$project: {
    title: "$primaryTitle",
    year: "$startYear",
    rating: "$rating.average",
    votes: "$rating.votes",
    directors: "$directorDetails.primaryName",
    actors: "$actorDetails.primaryName",
    genres: 1
}
}
])

```

Resultado esperado:

```
{
    "_id": "tt0111161",
    "title": "The Shawshank Redemption",
    "year": 1994,
    "rating": 9.3,
    "votes": 2500000,
    "directors": ["Frank Darabont"],
    "actors": ["Tim Robbins", "Morgan Freeman", ...],
    "genres": ["Drama"]
}
```

Índices usados: `idx_title_text_search` , `idx_people_id`

Consulta 2: Todas las Películas de un Director

Objetivo: Encontrar todas las películas dirigidas por un director específico.

```

db.titles.aggregate([
  {
    $lookup: {
      from: "people",
      localField: "crew.directors",
      foreignField: "_id",
      as: "directorInfo"
    }
  },
  {
    $match: {
      "directorInfo.primaryName": "Christopher Nolan"
    }
  },
  {
    $project: {
      title: "$primaryTitle",
      year: "$startYear",
      rating: "$rating.average",
      genres: 1
    }
  },
  {
    $sort: { startYear: -1 }
  }
])

```

Resultado esperado:

```
[
  {
    "title": "Oppenheimer",
    "year": 2023,
    "rating": 8.4,
    "genres": ["Biography", "Drama", "History"]
  },
  {
    "title": "Tenet",
    "year": 2020,
    "rating": 7.3,
    "genres": ["Action", "Sci-Fi"]
  }
]
```

```
},
// ... más películas
]
```

Índices usados: `idx_directors`, `idx_people_primaryName`

Consulta 3: Top 10 Películas por Rating

Objetivo: Obtener las 10 películas mejor valoradas (con mínimo de 100K votos).

```
db.titles.aggregate([
{
  $match: {
    titleType: "movie",
    "rating.votes": { $gte: 100000 }
  }
},
{
  $sort: {
    "rating.average": -1
  }
},
{
  $limit: 10
},
{
  $lookup: {
    from: "people",
    localField: "crew.directors",
    foreignField: "_id",
    as: "directorDetails"
  }
},
{
  $project: {
    title: "$primaryTitle",
    year: "$startYear",
    rating: "$rating.average",
    votes: "$rating.votes",
    directors: "$directorDetails.primaryName",
    genres: 1
  }
}
```

```
    }
])
```

Resultado esperado:

```
[
{
  "title": "The Shawshank Redemption",
  "year": 1994,
  "rating": 9.3,
  "votes": 2500000,
  "directors": ["Frank Darabont"],
  "genres": ["Drama"]
},
{
  "title": "The Godfather",
  "year": 1972,
  "rating": 9.2,
  "votes": 1800000,
  "directors": ["Francis Ford Coppola"],
  "genres": ["Crime", "Drama"]
},
// ... 8 más
]
```

Índices usados: `idx_type_rating_desc`

Consulta 4: Director Más Productivo

Objetivo: Encontrar el director que ha dirigido más películas.

```
db.titles.aggregate([
  {
    $match: {
      titleType: "movie",
      "crew.directors": { $exists: true, $ne: [] }
    }
  },
  {
    $unwind: "$crew.directors"
  },
  {
    $group: {
```

```

        _id: "$crew.directors",
        movieCount: { $sum: 1 }
    }
},
{
    $sort: { movieCount: -1 }
},
{
    $limit: 1
},
{
    $lookup: {
        from: "people",
        localField: "_id",
        foreignField: "_id",
        as: "directorInfo"
    }
},
{
    $project: {
        directorName: { $arrayElemAt: ["$directorInfo.primaryName", 0] },
        movieCount: 1
    }
}
])
)

```

Resultado esperado:

```
{
    "_id": "nm0001104",
    "directorName": "Steven Spielberg",
    "movieCount": 187
}
```

Índices usados: `idx_directors` , `idx_people_id`

Consulta 5: Top 10 Actores Más Prolíficos

Objetivo: Encontrar los actores que han participado en más películas.

```

db.titles.aggregate([
{
    $match: {

```

```

        "titleType": "movie",
        "principals.category": "actor"
    }
},
{
    $unwind: "$principals"
},
{
    $match: {
        "principals.category": "actor"
    }
},
{
    $group: {
        _id: "$principals.nconst",
        movieCount: { $sum: 1 }
    }
},
{
    $sort: { movieCount: -1 }
},
{
    $limit: 10
},
{
    $lookup: {
        from: "people",
        localField: "_id",
        foreignField: "_id",
        as: "actorInfo"
    }
},
{
    $project: {
        actorName: { $arrayElemAt: ["$actorInfo.primaryName", 0] },
        movieCount: 1
    }
}
])

```

Resultado esperado:

```
[
  {
    "_id": "nm0000151",
    "actorName": "Morgan Freeman",
    "movieCount": 142
  },
  {
    "_id": "nm0000234",
    "actorName": "Charlton Heston",
    "movieCount": 128
  },
  // ... 8 más
]
```

Índices usados: `idx_principals_category` , `idx_people_id`

Guía de Ejecución

Requisitos Previos

- Docker y Docker Compose instalados
- 30 GB de espacio en disco libre
- 8 GB de RAM (mínimo 4 GB)
- Archivos TSV de IMDb descargados

Paso 1: Descargar Dataset IMDb

```
# Descargar archivos desde https://datasets.imdbws.com/wget https://datasets.imdbws.co
m/name.basics.tsv.gz
wget https://datasets.imdbws.com/title.basics.tsv.gz
wget https://datasets.imdbws.com/title.akas.tsv.gz
wget https://datasets.imdbws.com/title.ratings.tsv.gz
wget https://datasets.imdbws.com/title.crew.tsv.gz
wget https://datasets.imdbws.com/title.principals.tsv.gz
wget https://datasets.imdbws.com/title.episode.tsv.gz
# Descomprimir todos los archivos gunzip *.gz
# Mover archivos a la carpeta data/mv *.tsv ./data/
```

Paso 2: Estructura del Proyecto

```
SDB2_PF2_Grupo10/
├── Fase3/
│   ├── docker-compose.yml
│   └── app/
│       ├── Dockerfile
│       ├── requirements.txt
│       └── loader.py
└── data/
    ├── name.basics.tsv
    ├── title.basics.tsv
    ├── title.akas.tsv
    ├── title.ratings.tsv
    ├── title.crew.tsv
    ├── title.principals.tsv
    └── title.episode.tsv
└── README.md
```

Paso 3: Levantar MongoDB con Docker

```
cd Fase3
docker-compose up -d --build
```

Paso 4: Ejecutar el Proceso ETL

```
docker-compose exec loader-app /bin/bash
python loader.py
```

Salida esperada durante la ejecución

```
[INFO] =====
[INFO] Iniciando proceso ETL de IMDb a MongoDB
[INFO] =====

[INFO] Fase 1: Carga Directa
[INFO] Cargando name.basics.tsv...
[INFO] 14,747,466 documentos insertados en 'people'
[INFO] Tiempo: 12 minutos

[INFO] Cargando title.basics.tsv...
[INFO] 11,813,847 documentos insertados en 'titles'
```

[INFO] Tiempo: 8 minutos

[INFO] Cargando title.akas.tsv...

[INFO] 40,000,000 documentos insertados en 'akas'

[INFO] Tiempo: 18 minutos

[INFO] Fase 2: Carga a Colecciones Temporales

[INFO] Cargando temp_ratings...

[INFO] 1,600,000 documentos insertados

[INFO] Cargando temp_crew...

[INFO] 11,813,847 documentos insertados

[INFO] Cargando temp_principals...

[INFO] 95,000,000 documentos insertados

[INFO] Cargando temp_episodes...

[INFO] 9,000,000 documentos insertados

[INFO] Tiempo: 25 minutos

[INFO] Fase 3: Merge (Embebido en titles)

[INFO] Mergeando ratings...

[INFO] 1,600,000 ratings embebidos - Tiempo: 3 min

[INFO] Mergeando crew...

[INFO] 11,813,847 crews embebidos - Tiempo: 5 min

[INFO] Mergeando principals...

[INFO] 95,000,000 principals procesados - Tiempo: 15 min

[INFO] Mergeando episodes...

[INFO] 9,000,000 episodios procesados - Tiempo: 4 min

[INFO] Fase 4: Limpieza

[INFO] Eliminando colecciones temporales...

[INFO] Limpieza completada

[INFO] =====

[INFO] ETL completado exitosamente

[INFO] Tiempo total: 90 minutos

[INFO] =====

[INFO] Creando índices en MongoDB...

[INFO] Índice idx_type_rating_desc creado en titles

[INFO] Índice idx_title_text_search creado en titles

[INFO] Índice idx_principals_category creado en titles

[INFO] Índice idx_directors creado en titles

[INFO] Índice idx_principals_nconst creado en titles

```
[INFO] Índice idx_people_primaryName creado en people
[INFO] Índice idx_akas_title creado en akas
[INFO] =====
[INFO] Todos los índices creados exitosamente
[INFO] Tiempo total: 8 minutos
[INFO] =====
```

Tiempo de Ejecución del ETL

Fase	Tiempo
Carga directa (people, titles base, akas)	~38 minutos
Carga temporal (ratings, crew, principals, episodes)	~25 minutos
Merge (embebido en titles)	~27 minutos
Limpieza	~2 minutos
TOTAL ETL	~92 minutos
Creación de índices	~8 minutos
TOTAL COMPLETO	~100 minutos

Decisiones Clave de Diseño

Embebido estratégico:

- Ratings, crew, principals y episodes embebidos en `titles`
- Elimina 4-5 JOINs por consulta típica
- Acelera lecturas dramáticamente

Referencias inteligentes:

- Colección `people` separada para evitar duplicar 14.7M nombres
- Actualizaciones centralizadas
- Ahorro significativo de espacio

Separación de datos auxiliares:

- Colección `akas` separada para 40M traducciones
- Mantiene documentos principales ligeros
- Carga bajo demanda

Tecnologías y Optimizaciones

- **Multiprocessing:** Reduce tiempo de carga 4x
- **Chunk processing:** Uso constante de memoria (~2GB)
- **Bulk operations:** 96x más rápido en operaciones de merge
- **Índices estratégicos:** 7 índices optimizados para patrones de consulta

Lecciones Aprendidas

1. **Diseñar basándose en consultas reales:** Los índices y el esquema deben optimizar las queries más frecuentes
2. **Embeber datos relacionados 1:1:** Si siempre se consultan juntos, deben estar juntos
3. **Referenciar datos que se repiten millones de veces:** No duplicar innecesariamente
4. **Usar bulk operations para escrituras masivas:** Son órdenes de magnitud más rápidas
5. **Indexar estratégicamente:** Balance entre velocidad de lectura y escritura

Referencias

- **Dataset IMDb:** <https://datasets.imdbws.com/>
- **MongoDB Documentation:** <https://docs.mongodb.com/>
- **PyMongo Documentation:** <https://pymongo.readthedocs.io/>
- **Docker Documentation:** <https://docs.docker.com/>
- **Pandas Documentation:** <https://pandas.pydata.org/docs/>

Proyecto desarrollado para: Sistema de Bases de Datos 2

Repositorio: https://github.com/MMP119/SDB2_PF2_Grupo10

Fase: 3 - Migración a MongoDB