



Grado en Ingeniería informática – Ingeniería del software
Programación de Bases de Datos
Curso 2023-2024

Proyecto PBD

Carlos García Rodríguez

DNI: 76051168N

William Adzet Mauchline

DNI: 42264667M

Índice

Índice	2
Introducción a los Modelos de Datos	3
Modelo de Datos en Bases de Datos Documentales	3
Modelo de Datos en Bases de Datos Clave-valor	4
Introducción a MongoDB	5
Operaciones de Lectura en MongoDB	5
Operaciones de Escritura en MongoDB	6
Indexación en MongoDB	7
Proyecto en MongoDB	7
Introducción a Riak	18
Operaciones de Lectura en Riak	19
Operaciones de escritura en Riak	20
Indexación en Riak	20
Proyecto en Riak	21
Bibliografía	35

Introducción a los Modelos de Datos

En primer lugar, se va a exponer que es un modelo de datos y para que se utiliza. Los modelos de datos se utilizan como **esquemas organizativos** que permiten mostrar la **estructura lógica de la base**, así como las **relaciones** y **limitaciones** que determinan cómo se almacenan los datos y cómo se puede acceder a ellos. Un modelo de datos también define que tipo de operaciones se pueden realizar con los datos y cómo se manipulan esos datos. Existen distintos tipos de modelos de datos, pero se va a entrar en detalle en las bases de datos no relacionales, para ello se van a enumerar los distintos tipos que existen pero se van a entrar en detalle en los documentales (**MongoDB**) y los de clave-valor (**Riak**).

- Clave-valor
- Documentos
- Gráficos
- En memoria

Modelo de Datos en Bases de Datos Documentales

Para poder explicar este modelo de datos es necesario entender el concepto de base de datos documental, se trata de una de las principales variantes de las **Bases de Datos NoSQL (no relacionales)**. Estas se caracterizan, como su propio nombre indica, en utilizan documentos para el almacenamiento de registros y los datos asociados a ellos. Cada uno de estos registros pueden almacenar distintos tipos de datos. Los **documentos** pueden tener distintos formatos, pero los más utilizados son **JSON**.

Las **Bases de Datos Documentales** son capaces de almacenar información en diferentes formatos sin una **estructura definida**. Aunque la estructura es distinta, estas bases de datos permiten realizar las mismas **operaciones básicas** que las **Bases de Datos Relaciones**, como por ejemplo, **insertar**, **actualizar**, **eliminar** datos.

Lo que las diferencia de las **Bases de Datos Relacionales** es que en las bases de datos orientadas a documentos no es necesario que se recorran todas las columnas de una tabla cuando se quiere realizar una consulta a una tabla, sino que en su lugar se asigna un identificador único a cada documento, de manera que cuando se quiere hacer una consulta comprueba el mismo documento.

Por último se van a detallar las **ventajas** y **desventajas** que tiene este tipo de modelo de datos. En cuanto a las ventajas:

- Son un modelo flexible que puede almacenar una gran cantidad de tipos de datos.
- Permiten almacenar y consultar información sin una estructura claramente definida.
- Aseguran una escritura rápida, dando prioridad a la disponibilidad de la escritura sobre la consistencia de los datos.
- Tienen una gran escalabilidad y son el mejor modelo de datos para almacenar grandes volúmenes de datos.
- Garantizan un buen rendimiento. La mayoría de bases de datos documentales cuentan con potentes motores de búsqueda y avanzadas propiedades de indexación.

En cuanto a las desventajas:

- No siempre pueden garantizar las propiedades **ACID** (Atomicidad, consistencia, integridad y durabilidad).
- No utilizan el lenguaje **SQL** como lenguaje principal de consulta.
- Los índices pueden llegar a ocupar mucha memoria **RAM**.
- No es fácil encontrar gran cantidad de información acerca de estas bases de datos.

Modelo de Datos en Bases de Datos Clave-valor

Se trata de un tipo de **Base de Datos NoSQL**, de igual manera que las bases de datos **documentales**, que funciona con un modelo simple de **clave** y **valor**. De manera que los datos se almacenan como pares **clave-valor**. La clave puede ser autogenerada y la única restricción que tiene es que tiene que ser única. En cambio, los valores tienen una estructura simple que acepta todo tipo de formatos.

Otra característica importante de este tipo de bases de datos es que utilizan **diccionarios** para clasificar y almacenar datos. Son fáciles de **escalar en sentido horizontal** y poseen una **gran velocidad** a la hora de hacer consultas a la base de datos.

A continuación, se van a detallar las **ventajas** y **desventajas** que tiene este tipo de modelo de datos. En cuanto a las ventajas:

- La estructura básica de clave-valor es muy simple, lo que favorece la implementación y la eficiencia en las operaciones de lectura y escritura, por lo que tiene un rendimiento rápido.
- Como se ha comentado anteriormente, son escalables horizontalmente, de manera que pueden distribuirse fácilmente en múltiples servidores. Esto permite que se puedan manejar grandes volúmenes de datos.
- Tienen una gran flexibilidad de tipos de datos, pudiendo manejar tanto datos estructurados como no estructurados.
- Como tiene una estructura tan simple, no necesitan una gran cantidad de administración y ajustes.

En cuanto a las desventajas:

- No son una gran opción para consultas complejas que tengan operaciones de filtrados y agrupaciones avanzadas. Esto puede resultar en limitaciones de expresividad de las consultas
- No son la mejor opción para modelos de datos altamente relaciones ya que la representación de relaciones complejas entre datos pueden resultar un problema.
- Actualizar un campo específico en un valor puede ser más complicado que con otros tipos de bases de datos, esto puede afectar a la eficiencia en actualizaciones de datos.
- La eficiencia de las consultas puede depender en gran medida de la efectividad de las claves de acceso elegidas.

Introducción a MongoDB

MongoDB se categoriza como una **Base de Datos NoSQL**. Se dice de **Bases de Datos NoSQL** aquellas que difieren de las bases de datos relaciones tradicionales basadas en **SQL**, estas **Bases de Datos** se han diseñado para poder solucionar limitaciones específicas de las bases de datos relacionales, por lo que pueden proporcionar soluciones más **flexibles** y **escalables** para algunos tipos de aplicaciones.

Una vez explicada que es una base de datos **NoSQL**, destacar que **MongoDB** es conocido como una solución de almacenamiento de datos orientado a documentos. Esto permite que en lugar de utilizar tablas y filas como en las **Bases de Datos** relacionales tradicionales, **MongoDB** almacena la información en documentos en formato **JSON**.

En el entorno de **MongoDB**, un **documento** es una estructura de datos **flexible** que puede contener múltiples campos, cada uno con su propio tipo de datos como en las **Bases de Datos** relacionales. Esta **flexibilidad** en el esquema permite al usuario poder almacenar datos de manera más **dinámica** y con una mayor **adaptabilidad** si se compara con las **Bases de Datos** relaciones. No es necesario definir un esquema fijo, sino que facilita la incorporación de nuevos campos y la modificación de la estructura de los documentos en función de las necesidades que tenga el proyecto.

A continuación, se van a presentar dos de las más importantes características que tiene **MongoDB**:

- **Flexibilidad del Esquema:** **MongoDB** tiene un esquema **flexible**, donde las colecciones no obligan al **SGBD** a tener una estructura idéntica para todos los documentos. Esto permite que los documentos de la misma colección no tengan obligatoriamente el mismo número de campos o la misma estructura. Cada documento solo necesita el número de campos que necesite de la entidad o documento que representa.
- **Escalabilidad Horizontal:** Esta característica permite poder tener los datos en varios servidores, de esta manera se mejora el rendimiento y la capacidad de manejar grandes volúmenes de datos o información. Que sea **escalabilidad horizontal** facilita la expansión de los recursos sin la necesidad de verse obligado a cambiar el **hardware** a **hardware** más potente.

Operaciones de Lectura en MongoDB

Las operaciones más importantes de lectura en MongoDB son “**find**” y “**find_one**”. Estas operaciones permiten recuperar datos, la manera en la que están definidos los campos permite que exista una optimización de consultas bastante importante, de modo que tiene la capacidad de seleccionar y devolver solo un subconjunto específico de campos de un documento, en lugar de recuperar todo el documento entero. Esta sería la estructura de ambas operaciones:

- **db.<collection>.find():** Imprime los primeros 20 documentos que encuentra (por defecto), pero se puede cambiar el número de documentos que imprime utilizando **.limit(n)**
 - **db.<collection>.find({"clave" : "valor"}, {"clave" : "valor"}):** Imprime los documentos que cumplan la condición de clave-valor.
 - **db.<collection>.find({"clave" : {\$gt: "valor"}}):** Imprime los documentos que tengan el valor de la clave mayor del especificado. Puede cambiarse a mayor-igual, menor y menor-igual.
 - **\$gte**
 - **\$lte**
 - **\$gt**
 - **\$lt**

- **db.<collection>.findOne()**: Imprime solo el primer documento que encuentra.

Operaciones de Escritura en MongoDB

Las operaciones más importantes de escritura en MongoDB son “**insert**”, “**update**” y “**delete**”. Estas operaciones permiten al usuario poder manipular documentos en las colecciones. Estas son sus estructuras:

- **db.<collection>.insert({ campo1 : valor1 }, { campo2 : valor2 })**: Inserta un documento o varios documentos en una colección.
 - **db.<collection>.insertMany([{ campo1 : valor1, campo2 : valor2 }, { campo3 : valor3, campo4 : valor4 }])**: Inserta varios documentos en una colección
- **db.<collection>.update({ campo : valor }, { \$set: { campoNuevo : valorNuevo } })**: Modifica un documento o documentos en una colección. El método puede modificar campos específicos de un documento o reemplazar un documento por completo.
 - **db.<collection>.updateMany({ campo : valor }, { \$set: { campoNuevo : valorNuevo } })**: Actualiza varios documentos que cumplen con la condición (filtro).
 - **db.<collection>.replaceOne({ _id : ObjectId("idDocumento") }, { campoNuevo : valorNuevo })**: Reemplaza un documento completo, pero conservado el “_id”.
- **db.<collection>.deleteOne(<filter>)**: Elimina un solo documento que cumpla con el filtro de una colección.
 - **db.<collection>.deleteMany(<filter>)**: Elimina todos los documentos que cumpla con el filtro de una colección.

Antes de pasar al siguiente punto, se van a exponer algunos conceptos que pueden resultar interesantes sobre las operaciones de escritura en MongoDB, los **Upserts**:

Los **Upserts** son una combinación de un “**update**” y un “**insert**”. Esto permite realizar actualización en caso que exista un documento que cumpla el criterio que se ha impuesto (**filtro**) y en caso que no cumpla con el criterio, se inserta como un documento nuevo. Esta sería la estructura que sigue:

- **db.<collection>.update({ campo : valor }, { \$set: { campoNuevo : valorNuevo } }, { upsert:true })**

Indexación en MongoDB

La **indexación** es un mecanismo que permite mejorar el **rendimiento** de las consultas en las **Bases de Datos**, en este caso, en **MongoDB**. Esto permite que **MongoDB** no necesite recorrer toda la colección, sino que haga uso de los índices para poder realizar la búsqueda de una forma más eficiente.

Un **índice** se trata como una estructura de datos especiales que almacena una pequeña porción de datos de la colección de tal forma que sea más fácil de recorrer. El **índice** almacena el valor de un campo específico o de un conjunto de campos, ordenados por el valor de dichos campos. Estos

mejoran la velocidad de **búsqueda** y recuperación de datos al proporcionar un acceso mucho más rápido a los datos.

MongoDB ofrece una amplia gama de tipos de **índices** y funciones con sus criterios de clasificación específicos del idioma para admitir patrones de **acceso complejos** a sus datos. Los índices se pueden crear y eliminar según se quiera para poder adaptarse a los requisitos que la aplicación tenga. También permite que se puedan declarar en cualquier campo dentro de sus documentos. Algunos de esos **índices** son:

- **Índices simples:** Se crean con un único campo y mejoran la eficiencia de búsquedas en ese campo en específico.
- **Índices compuestos:** Se crean en varios campos por lo que son útiles para consultas que tengan varios criterios de búsqueda.
- **Índices texturales:** Se utilizan para mejorar la búsqueda de texto completo para aquellos campos que son de tipo texto.
- **Índices geoespaciales:** Están optimizados para aquellas consultas que contengan datos geoespaciales, como coordenadas.

La creación de índices sigue la siguiente estructura:

- **db.<coleccion>.createIndex({ campo : 1 }):** Para la creación de un índice simple.
- **db.<coleccion>.createIndex({ campo1 : 1, campo2 : -1 }):** Para la creación de un índice compuesto.

MongoDB también permite la gestión de **índices**, para poder modificar o eliminar índices según sea necesario. Se puede utilizar el método “**explain()**”, este método devuelve información sobre el plan de consulta por lo que se puede utilizar para analizar y optimizar el rendimiento de las consultas.

Proyecto en MongoDB

Después de haber expuesto la teoría de **MongoDB**, se van a exponer distintos ejercicios (**consultas**) sobre una **Base de Datos** que se ha creado en **MongoDB** y la posterior inserción de datos desde un archivo “**.csv**” descargado de la página: <https://www.kaggle.com/> . El proyecto se ha realizado con la herramienta **Visual Studio Code**.

Este sería el código correspondiente:

```
import pymongo
import csv
import pandas as pd
import re
from datetime import datetime

client = pymongo.MongoClient("mongodb://localhost:27017")
db = client["Movies_MongoDB"]
```

```

collection = db["ratedMovies"]
collection_name = "ratedMovies"

def collection_exists(db, collection_name):
    return collection_name in db.list_collection_names()

def collection_data(collection):
    return collection.count_documents({}) > 0

if collection_exists(db, collection_name) and collection_data:
    print(f"La colección '{collection_name}' ya existe y tiene datos, por lo que no se insertará")
else:
    datos_movie = pd.read_csv("Top_rated_movies1.csv", dtype={"title" : str, "overview": str, "popularity": float, "release_date": str, "vote_average": float, "vote_count": int})
    collection.insert_many(datos_movie.to_dict("records"))

def pelicula_por_titulo(title):
    regex_pattern = re.compile(f'.*{re.escape(title)}.*', re.IGNORECASE)
    resultados = collection.find({'title': {'$regex': regex_pattern}})
    return resultados

def pelicula_por_fecha(year):
    resultados = collection.find({"release_date": {"$regex": f"^{year}"}}), {"_id": 0})
    return resultados

def pelicula_por_votos(votes):
    resultados = collection.find({'vote_average': {'$gte': float(votes)}})
    return resultados

def pelicula_por_popularidad(popularity):
    resultados = collection.find({'popularity': {'$gte': float(popularity)}})
    return resultados

def mostrar_resultados(resultados):
    for resultado in resultados:
        print(resultado)

```



```

while True:
    print("Menú de consultas")
    print("1. Consulta por título")
    print("2. Consulta por fecha")
    print("3. Consulta por votos")
    print("4. Consulta por popularidad")
    print("5. Salir")

    opcion = input("Seleccione el tipo de consulta que le gustaría
hacer (1-5): ")

    if opcion == "1":
        title = input("Inserte el título: ")
        resultados = pelicula_por_titulo(title)
        mostrar_resultados(resultados)
    elif opcion == "2":
        year = input("Inserte el año: ")
        resultados = pelicula_por_fecha(year)
        mostrar_resultados(resultados)
    elif opcion == "3":
        votes = input("Inserte el número mínimo de la media de votos:
")
        resultados = pelicula_por_votos(votes)
        mostrar_resultados(resultados)
    elif opcion == "4":
        popularity = input("Inserte el número mínimo de popularidad: ")
        resultados = pelicula_por_popularidad(popularity)
        mostrar_resultados(resultados)
    elif opcion == "5":
        print("Saliendo del programa")
        break
    else:
        print("La opción que ha seleccionado no es válida")

client.close()

```

En primer lugar, comentar que es muy importante realizar los siguientes pasos antes de ejecutar el proyecto, abrir una nueva terminal y movernos a la carpeta donde está situada el proyecto y posteriormente escribir en la terminal “**pip install pymongo**” y “**pip install pandas**” para poder utilizar la librería de **Python** para **MongoDB** y para poder utilizar la librería de **Pandas**. Una vez

tengamos Pandas y de MongoDB se puede ejecutar el proyecto utilizando el siguiente comando “python ProyectoPBDMongo.py”

```
PS C:\Users\carlo> cd C:\ProyectoPBD\ProyectoMongo
PS C:\ProyectoPBD\ProyectoMongo> python ProyectoPBDMongo.py
La colección 'ratedMovies ya existe y tiene datos, por lo que no se insertará
Menú de consultas
1. Consulta por titulo
2. Consulta por fecha
3. Consulta por votos
4. Consulta por popularidad
5. Salir
Seleccione el tipo de consulta que le gustaría hacer (1-5):
```

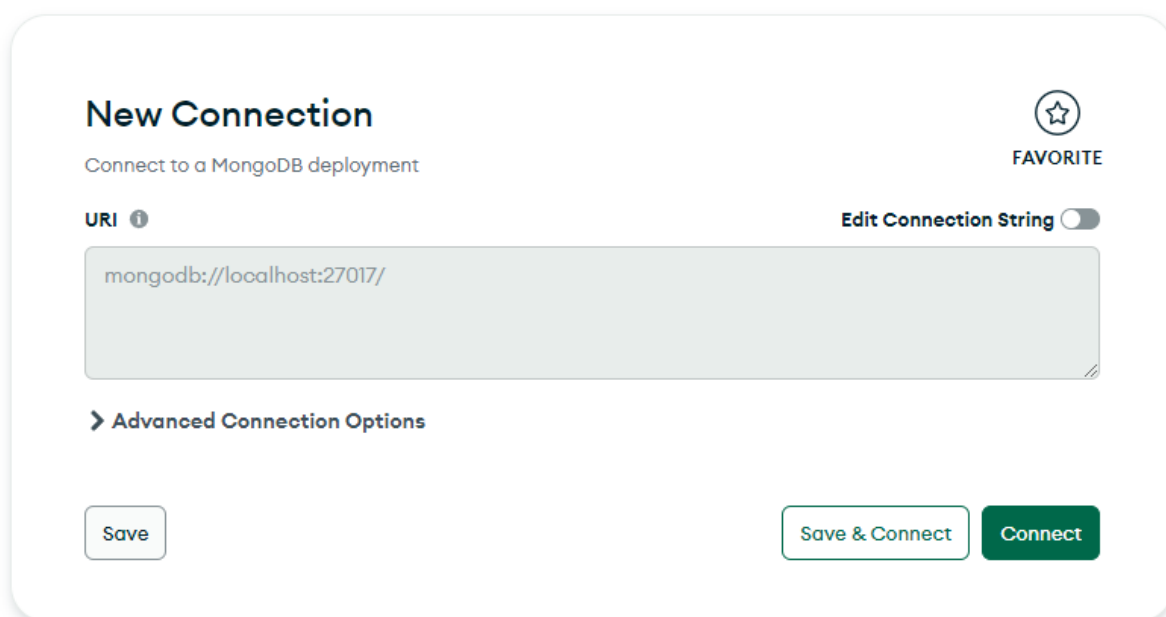
A continuación se va a explicar el código detalladamente:

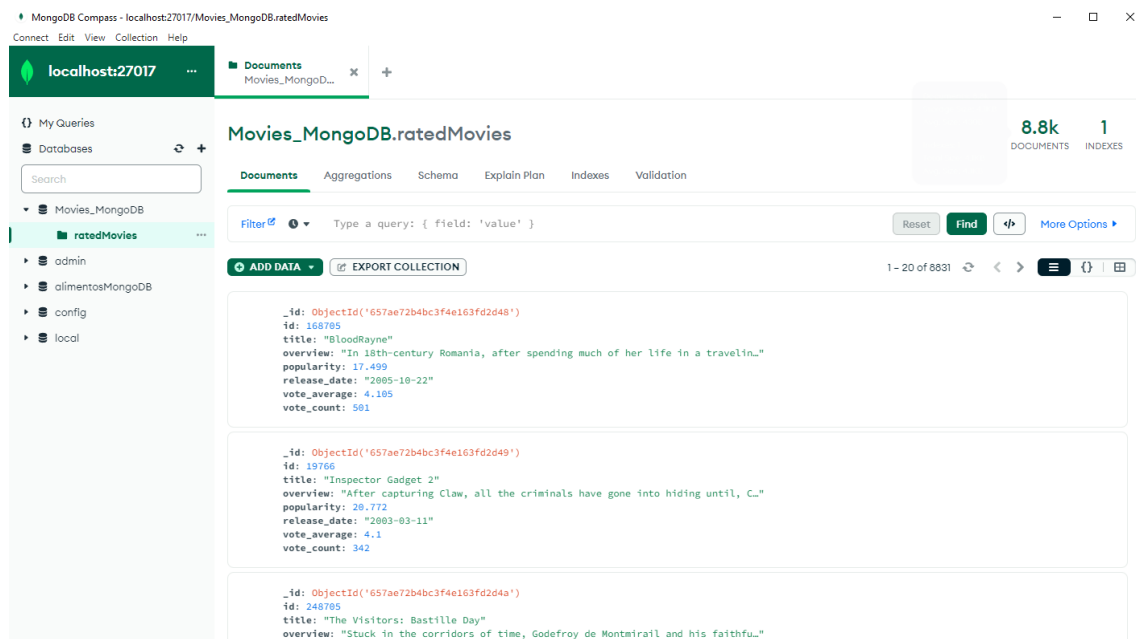
Se importan las librerías necesarias para el proyecto: **pymongo**, **csv**, **pandas** con un alias “**pd**”, **re** y **datetime**. Cada una de ellas se irán explicando según se vaya viendo el código que las utiliza.

A continuación se crea una variable denominada “**client**” que permitirá realizar la conexión con la **Base de Datos MongoDB** (**importante**: poner **localhost** y el puerto asignado, en este caso “**27017**”). Se crea la propia **Base de Datos** denominada “**Movies_MongoDB**” y por último la colección dentro de la **Base de Datos** denominada “**ratedMovies**”.

Se almacena en “**datos_movie**”, todos los datos del archivo “.csv” y se inserta en la **Base de Datos** mediante la instrucción “**collection.insert_many(datos_movie.to_dict(“records”))**”

Una vez ya están insertados todos los datos en la **Base de Datos de MongoDB** se puede proceder a realizar las consultas de los datos. (Se puede comprobar que se han insertado los datos correctamente utilizando la herramienta **MongoDB Compass** que simula una interfaz gráfica donde poder comprobar todas las **Bases de Datos**, con sus respectivas colecciones y sus respectivos datos).





Como se puede observar, se han insertado todos los datos correctamente.

A continuación se expondrán algunas consultas que hemos seleccionado para la realización del proyecto:

- En primer lugar, se hará una consulta por el título de la película, esta consulta se ha implementado de manera que no sea necesario escribir el nombre completo entero, sino que exista la cadena que se ha introducido por consola en el título de la película. Para ello se ha implementado el siguiente método:

```
def pelicula_por_titulo(title):
    regex_pattern = re.compile(f'.*{re.escape(title)}.*',
re.IGNORECASE)
    resultados = collection.find({'title': {'$regex':
regex_pattern}})
    return resultados
```

Se ha hecho uso de una expresión regular para poder filtrar aquellos títulos que cumplen con la cadena de caracteres introducida por el usuario.

La salida por consola sería la siguiente:

Menú de consultas

1. Consulta por título
2. Consulta por fecha
3. Consulta por votos
4. Consulta por popularidad
5. Salir

Seleccione el tipo de consulta que le gustaría hacer (1-5): 1

Inserte el título: nado

```
{'_id': ObjectId('657aefb4e2997e9056102213'), 'id': 205321, 'title': 'Sharknado', 'overview': 'A freak hurricane hits Los Angeles, causing man-eating sharks to be scooped up in tornadoes and flooding the city with shark-infested seawater. Surfer and bar-owner Fin sets out with his friends Baz and Nova to rescue his estranged wife April and teenage daughter Claudia.', 'popularity': 13.862, 'release_date': '2013-07-11', 'vote_average': 3.917, 'vote_count': 1482}
{'_id': ObjectId('657aefb4e2997e9056104409'), 'id': 438970, 'title': 'Sharknado 5: Global Swarming', 'overview': 'Fin and his wife April travel around the world to save their young son who's trapped inside a sharknado.', 'popularity': 16.51, 'release_date': '2017-08-06', 'vote_average': 4.779, 'vote_count': 303}
{'_id': ObjectId('657aefb4e2997e9056104428'), 'id': 331446, 'title': 'Sharknado 3: Oh Hell No!', 'overview': 'The sharks take bite out of the East Coast when the sharknado hits Washington, D.C. and Orlando, Florida.', 'popularity': 17.707, 'release_date': '2015-07-22', 'vote_average': 4.635, 'vote_count': 466}
{'_id': ObjectId('657aefb4e2997e9056104431'), 'id': 248504, 'title': 'Sharknado 2: The Second One', 'overview': 'A freak weather system turns its deadly fury on New York City, unleashing a Sharknado on the population and its most cherished, iconic sites - and only Fin and April can save the Big Apple.', 'popularity': 19.079, 'release_date': '2014-07-31', 'vote_average': 4.584, 'vote_count': 611}
{'_id': ObjectId('657aefb4e2997e9056104435'), 'id': 390989, 'title': 'Sharknado 4: The 4th Awakens', 'overview': 'The new installment of the Sharknado franchise takes place 5 years after Sharknado 3: Oh Hell No! There have been no Sharknados in the intervening years, but now they're appearing again in unexpected ways.', 'popularity': 17.537, 'release_date': '2016-07-31', 'vote_average': 4.6, 'vote_count': 354}
```

Como se puede observar, se han mostrado todas las tuplas con sus respectivos datos que cumplen con la condición de tener “**nado**” en el título.

- En segundo lugar una consulta por una fecha en concreto, donde el usuario pueda buscar películas que se hayan estrenado en un año en concreto. De igual manera que el método anterior se ha hecho uso de una expresión regular para poder utilizar solo el año. Para ello se ha implementado el siguiente método:

```
def pelicula_por_fecha(year):
    resultados = collection.find({"release_date": {"$regex": f"^{year}"}} , {"_id": 0})
    return resultados
```

Se ha utilizado el campo “**release_date**” y se ha extraído el año para poder comparar con el año que ha introducido el usuario por consola.

La salida por consola sería la siguiente:

Menú de consultas

1. Consulta por titulo
2. Consulta por fecha
3. Consulta por votos
4. Consulta por popularidad

5. Salir

Seleccione el tipo de consulta que le gustaría hacer (1-5): 2

Inserte el año: 2023

{'id': 1000492, 'title': 'All Your Faces', 'overview': 'Since 2014, France's restorative justice programmes have offered a safe space for supervised dialogue between offenders and victims. Grégoire, Nawelle, and Sabine, victims of heists and violent robberies, agree to join one of these discussion groups alongside offenders Nassim, Issa, and Thomas, all convicted of violent robberies. Meanwhile Chloé, a victim of childhood sexual abuse, prepares for dialogue with her own aggressor after learning he has moved back into town.', 'popularity': 21.525, 'release_date': '2023-03-29', 'vote_average': 8.107, 'vote_count': 304}

{'id': 1059377, 'title': 'Srimulat: Hidup Memang Komedi', 'overview': 'After Gepeng's indiscipline, Srimulat's life in the capital suddenly turned upside down. The story of love, career and generations of legendary comedians continues!'", 'popularity': 11.038, 'release_date': '2023-11-23', 'vote_average': 4.0, 'vote_count': 1}

{'id': 569094, 'title': 'Spider-Man: Across the Spider-Verse', 'overview': 'After reuniting with Gwen Stacy, Brooklyn's full-time, friendly neighborhood Spider-Man is catapulted across the Multiverse, where he encounters the

Spider Society, a team of Spider-People charged with protecting the Multiverse's very existence. But when the heroes clash on how to handle a new threat, Miles finds himself pitted against the other Spiders and must set out on his own to save those he loves most.', 'popularity': 448.515, 'release_date': '2023-05-31', 'vote_average': 8.403, 'vote_count': 4976}

{'id': 626332, 'title': 'Flamin' Hot', 'overview': 'The inspiring true story of Richard Montañez, the Frito Lay janitor who channeled his Mexican American heritage and upbringing to turn the iconic Flamin' Hot Cheetos into a

snack that disrupted the food industry and became a global pop culture phenomenon.', 'popularity': 103.072, 'release_date': '2023-03-11', 'vote_average': 8.236, 'vote_count': 432}

{'id': 872585, 'title': 'Oppenheimer', 'overview': 'The story of J. Robert Oppenheimer's role in the development of the atomic bomb during World War II.', 'popularity': 1766.305, 'release_date': '2023-07-19', 'vote_average': 8.163, 'vote_count': 4905}

.

.

.

{'id': 1063422, 'title': 'The Strays', 'overview': 'A Black woman's meticulously crafted life of privilege starts to unravel when two strangers show up in her quaint suburban town.', 'popularity': 33.593, 'release_date': '2023-02-17', 'vote_average': 5.492, 'vote_count': 308}

{'id': 980078, 'title': 'Winnie the Pooh: Blood and Honey', 'overview': 'Christopher Robin is headed off to college and he has abandoned his old friends, Pooh and Piglet, which then leads to the duo embracing their inner monsters.', 'popularity': 68.676, 'release_date': '2023-01-27', 'vote_average': 5.281, 'vote_count': 1038}

{'id': 643215, 'title': 'Asterix & Obelix: The Middle Kingdom', 'overview': 'Gallic heroes and forever friends Asterix and Obelix journey to China to help Princess Sa See save the Empress and her land from a nefarious prince.', 'popularity': 48.019, 'release_date': '2023-02-01', 'vote_average': 4.901, 'vote_count': 754}

Estas serían todas las películas que se estrenaron en el año **2023**(como el número de tuplas son muchas, se han mostrado solo algunas de ellas.

- En tercer lugar una consulta por número mínimo de la media de votos de una película, donde el usuario introducirá un valor que podrá ser decimal para poner un límite mínimo de media de valoración de una película. Para ello se ha implementado el siguiente método:

```
def pelicula_por_votos(votes):
    resultados = collection.find({'vote_average': {'$gte':
float(votes)}})
    return resultados
```

La salida por consola sería la siguiente:

Menú de consultas

1. Consulta por título
2. Consulta por fecha
3. Consulta por votos
4. Consulta por popularidad
5. Salir

Seleccione el tipo de consulta que le gustaría hacer (1-5): 3

Inserte el número mínimo de la media de votos: 8.5

{'_id': ObjectId('657aefb4e2997e905610227d'), 'id': 238, 'title': 'The Godfather', 'overview': 'Spanning the years 1945 to 1955, a chronicle of the fictional Italian-American Corleone crime family. When organized crime family patriarch, Vito Corleone barely survives an attempt on his life, his youngest son, Michael steps in to take care of the would-be killers, launching a campaign of bloody revenge.', 'popularity': 167.536, 'release_date': '1972-03-14', 'vote_average': 8.708, 'vote_count': 18991}

{'_id': ObjectId('657aefb4e2997e905610227e'), 'id': 278, 'title': 'The Shawshank Redemption', 'overview': 'Framed in the 1940s for the double murder of his wife and her lover, upstanding banker Andy Dufresne begins a new life at the Shawshank prison, where he puts his accounting skills to work for an amoral warden. During his long stretch in prison, Dufresne comes to be admired by the other inmates -- including an older prisoner named Red -- for his integrity and unquenchable sense of hope.', 'popularity': 179.959, 'release_date': '1994-09-23', 'vote_average': 8.705, 'vote_count': 24985}

{'_id': ObjectId('657aefb4e2997e905610227f'), 'id': 240, 'title': 'The Godfather Part II', 'overview': 'In the continuing saga of the Corleone crime family, a young Vito Corleone grows up in Sicily and in 1910s New York. In the 1950s, Michael Corleone attempts to expand the family business into Las Vegas, Hollywood and Cuba.', 'popularity': 106.688, 'release_date': '1974-12-20', 'vote_average': 8.589, 'vote_count': 11467}

{'_id': ObjectId('657aefb4e2997e9056102280'), 'id': 424, 'title': 'Schindler's List', 'overview': 'The true story of how businessman Oskar Schindler saved over a thousand Jewish lives from the Nazis while they worked as slaves in his factory during World War II.', 'popularity': 81.656, 'release_date': '1993-12-15', 'vote_average': 8.571, 'vote_count': 14813}

{'_id': ObjectId('657aefb4e2997e9056102281'), 'id': 19404, 'title': 'Dilwale Dulhania Le Jayenge', 'overview': 'Raj is a rich, carefree, happy-go-lucky second generation NRI. Simran is the daughter of Chaudhary Baldev Singh, who in spite of being an NRI is very strict about adherence to Indian values. Simran has left for India to be married to her childhood fiancé. Raj leaves for India with a mission at his hands, to claim his lady love under the noses of her whole family. Thus begins a saga.', 'popularity': 43.081, 'release_date': '1995-10-20', 'vote_average': 8.546, 'vote_count': 4282}

{'_id': ObjectId('657aefb4e2997e9056102282'), 'id': 389, 'title': '12 Angry Men', 'overview': 'The defense and the prosecution have rested and the jury is filing into the jury room to decide if a young Spanish-American is guilty or innocent of murdering his father. What begins as an open and shut case soon becomes a mini-drama of each of the jurors' prejudices and preconceptions about the trial, the accused, and each other.', 'popularity': 66.507, 'release_date': '1957-04-10', 'vote_average': 8.545, 'vote_count': 7805}

{'_id': ObjectId('657aefb4e2997e9056102283'), 'id': 129, 'title': 'Spirited Away', 'overview': 'A young girl, Chihiro, becomes trapped in a strange new world of spirits. When her parents undergo a mysterious transformation, she must call upon the courage she never knew she had to free her family.', 'popularity': 104.619, 'release_date': '2001-07-20', 'vote_average': 8.54, 'vote_count': 15131}

{'_id': ObjectId('657aefb4e2997e9056102284'), 'id': 496243, 'title': 'Parasite', 'overview': 'All unemployed, Ki-taek's family takes peculiar interest in the wealthy and glamorous Parks for their livelihood until they get entangled in an unexpected incident.', 'popularity': 106.749, 'release_date': '2019-05-30', 'vote_average': 8.515, 'vote_count': 16684}

{'_id': ObjectId('657aefb4e2997e9056102285'), 'id': 155, 'title': 'The Dark Knight', 'overview': 'Batman raises the stakes in his war on crime. With the help of Lt. Jim Gordon and District Attorney Harvey Dent, Batman sets out to dismantle the remaining criminal organizations that plague the streets. The partnership proves to be effective, but they soon find themselves prey to a reign of chaos unleashed by a rising criminal mastermind known to the terrified citizens of Gotham as the Joker.', 'popularity': 127.08, 'release_date': '2008-07-16', 'vote_average': 8.514, 'vote_count': 30958}

{'_id': ObjectId('657aefb4e2997e9056102286'), 'id': 497, 'title': 'The Green Mile', 'overview': 'A supernatural tale set on death row in a Southern prison, where gentle giant John Coffey possesses the mysterious power to heal people's ailments. When the cell block's head guard, Paul Edgecomb, recognizes Coffey's miraculous gift, he tries desperately to help stave off the condemned man's execution.', 'popularity': 100.097, 'release_date': '1999-12-10', 'vote_average': 8.509, 'vote_count': 16159}

{'_id': ObjectId('657aefb4e2997e9056102287'), 'id': 372058, 'title': 'Your Name.', 'overview': 'High schoolers Mitsuha and Taki are complete strangers living separate lives. But one night, they suddenly switch places. Mitsuha wakes up in Taki's body, and he in hers. This bizarre occurrence continues to happen randomly, and the two must adjust their lives around each other.', 'popularity': 112.597, 'release_date': '2016-08-26', 'vote_average': 8.505, 'vote_count': 10513}

Como se puede observar se han mostrado todos los títulos que superan la media de **8.5** de valoración.

- Para la cuarta y última consulta, se trata de una consulta por popularidad, de manera que el usuario podrá buscar una película que supere un mínimo de popularidad. Para ello se ha implementando el siguiente método:

```
def pelicula_por_popularidad(popularity):
    resultados = collection.find({'popularity': {'$gte':
float(popularity)}})
    return resultados
```

La salida por consola sería la siguiente:

Menú de consultas

1. Consulta por título
2. Consulta por fecha
3. Consulta por votos
4. Consulta por popularidad
5. Salir

Seleccione el tipo de consulta que le gustaría hacer (1-5): 4

Inserte el número mínimo de popularidad: 500.400

{'_id': ObjectId('657aefb4e2997e905610230f'), 'id': 872585, 'title': 'Oppenheimer', 'overview': 'The story of J. Robert Oppenheimer's role in the development of the atomic bomb during World War II.', 'popularity': 1766.305, 'release_date': '2023-07-19', 'vote_average': 8.163, 'vote_count': 4905}

{'_id': ObjectId('657aefb4e2997e905610242d'), 'id': 507089, 'title': 'Five Nights at Freddy's', 'overview': 'Recently fired and desperate for work, a troubled young man named Mike agrees to take a position as a night security guard at an abandoned theme restaurant: Freddy Fazbear's Pizzeria. But he soon discovers that nothing at Freddy's is what it seems.', 'popularity': 1252.998, 'release_date': '2023-10-25', 'vote_average': 7.888, 'vote_count': 2516}

{'_id': ObjectId('657aefb4e2997e90561024fd'), 'id': 502356, 'title': 'The Super Mario Bros. Movie', 'overview': 'While working underground to fix a water main, Brooklyn plumbers—and brothers—Mario and Luigi are transported down a mysterious pipe and wander into a magical new world. But when the brothers are separated, Mario embarks on an epic quest to find Luigi.', 'popularity': 502.499, 'release_date': '2023-04-05', 'vote_average': 7.75, 'vote_count': 7233}

{'_id': ObjectId('657aefb4e2997e9056102648'), 'id': 575264, 'title': 'Mission: Impossible - Dead Reckoning Part One', 'overview': 'Ethan Hunt and his IMF team embark on their most dangerous mission yet: To track down a terrifying new weapon that threatens all of humanity before it falls into the wrong hands. With control of the future and the world's fate at stake and dark forces from Ethan's past closing in, a deadly race around the globe begins. Confronted by a mysterious, all-powerful enemy, Ethan must consider that nothing can matter more than his mission—not even the lives of those he cares about most.', 'popularity': 762.172, 'release_date': '2023-07-08', 'vote_average': 7.586, 'vote_count': 2534}

{'_id': ObjectId('657aefb4e2997e905610283e'), 'id': 926393, 'title': 'The Equalizer 3', 'overview': 'Robert McCall finds himself at home in Southern Italy but he discovers his friends are under the control of local crime bosses. As events turn deadly, McCall knows what he has to do: become his friends' protector by taking on the mafia.', 'popularity': 573.141, 'release_date': '2023-08-30', 'vote_average': 7.42, 'vote_count': 1707}

{'_id': ObjectId('657aefb4e2997e9056102868'), 'id': 951491, 'title': 'Saw X', 'overview': 'Between the events of 'Saw' and 'Saw II', a sick and desperate John Kramer travels to Mexico for a risky and experimental medical procedure in hopes of a miracle cure for his cancer, only

to discover the entire operation is a scam to defraud the most vulnerable. Armed with a newfound purpose, the infamous serial killer returns to his work, turning the tables on the con artists in his signature visceral way through devious, deranged, and ingenious traps.", 'popularity': 509.974, 'release_date': '2023-09-26', 'vote_average': 7.413, 'vote_count': 1150}

{ '_id': ObjectId('657aefb4e2997e9056102a88'), 'id': 695721, 'title': 'The Hunger Games: The Ballad of Songbirds & Snakes', 'overview': '64 years before he becomes the tyrannical president of Panem, Coriolanus Snow sees a chance for a change in fortunes when he mentors Lucy Gray Baird, the female tribute from District 12.', 'popularity': 595.362, 'release_date': '2023-11-15', 'vote_average': 7.2, 'vote_count': 401}

{ '_id': ObjectId('657aefb4e2997e9056102b10'), 'id': 385687, 'title': 'Fast X', 'overview': "Over many missions and against impossible odds, Dom Toretto and his family have outsmarted, out-nerved and outdriven every foe in their path. Now, they confront the most lethal opponent they've ever faced: A terrifying threat emerging from the shadows of the past who's fueled by blood revenge, and who is determined to shatter this family and destroy everything—and everyone—that Dom loves, forever.", 'popularity': 749.903, 'release_date': '2023-05-17', 'vote_average': 7.208, 'vote_count': 4335}

{ '_id': ObjectId('657aefb4e2997e9056102bdf'), 'id': 670292, 'title': 'The Creator', 'overview': 'Amid a future war between the human race and the forces of artificial intelligence, a hardened ex-special forces agent grieving the disappearance of his wife, is recruited to hunt down and kill the Creator, the elusive architect of advanced AI who has developed a mysterious weapon with the power to end the war—and mankind itself.', 'popularity': 1622.502, 'release_date': '2023-09-27', 'vote_average': 7.156, 'vote_count': 1128}

{ '_id': ObjectId('657aefb4e2997e9056102bf0'), 'id': 670292, 'title': 'The Creator', 'overview': 'Amid a future war between the human race and the forces of artificial intelligence, a hardened ex-special forces agent grieving the disappearance of his wife, is recruited to hunt down and kill the Creator, the elusive architect of advanced AI who has developed a mysterious weapon with the power to end the war—and mankind itself.', 'popularity': 1622.502, 'release_date': '2023-09-27', 'vote_average': 7.149, 'vote_count': 1114}

{ '_id': ObjectId('657aefb4e2997e9056102f16'), 'id': 565770, 'title': 'Blue Beetle', 'overview': 'Recent college grad Jaime Reyes returns home full of aspirations for his future, only to find that home is not quite as he left it. As he searches to find his purpose in the world, fate intervenes when Jaime unexpectedly finds himself in possession of an ancient relic of alien biotechnology: the Scarab.', 'popularity': 515.987, 'release_date': '2023-08-16', 'vote_average': 6.9, 'vote_count': 1714}

{ '_id': ObjectId('657aefb4e2997e905610322c'), 'id': 615656, 'title': 'Meg 2: The Trench', 'overview': 'An exploratory dive into the deepest depths of the ocean of a daring research team spirals into chaos when a malevolent mining operation threatens their mission and forces them into a high-stakes battle for survival.', 'popularity': 503.411, 'release_date': '2023-08-02', 'vote_average': 6.745, 'vote_count': 2586}

{ '_id': ObjectId('657aefb4e2997e9056103756'), 'id': 299054, 'title': 'Expend4bles', 'overview': 'Armed with every weapon they can get their hands on and the skills to use them, The Expendables are the world's last line of defense and the team that gets called when all other options are off the table. But new team members with new styles and tactics are going to give "new blood" a whole new meaning.', 'popularity': 873.341, 'release_date': '2023-09-15', 'vote_average': 6.438, 'vote_count': 792}

Como se puede observar en la salida, aparecen todas las películas con una popularidad mínima de **500.400**

Introducción a Riak

Riak KV, también conocido como Riak Key-Value, es una base de datos NoSQL distribuida diseñada para proporcionar alta disponibilidad, tolerancia a fallos y escalabilidad.

Es desarrollado por Basho Technologies, ahora propiedad de Bet365. Riak KV se centra específicamente en almacenar y recuperar pares clave-valor de manera distribuida y tolerante a fallos.

- **Arquitectura Distribuida:** Riak KV está diseñado como una base de datos distribuida, lo que significa que **puede abarcar varios nodos en una red**. Esta distribución permite el **escalado horizontalmente** añadiendo nodos al clúster.
- **Tolerancia a fallos:** Riak KV está diseñado para ser tolerante a fallos. **Los datos se replican en varios nodos** y, en caso de fallo de un nodo, el sistema puede seguir operando sin problemas accediendo a copias de un nodo.
- **Alta Disponibilidad:** Debido a su naturaleza distribuida y a sus mecanismos de replicación, Riak KV proporciona alta disponibilidad. **Se puede acceder a la base de datos incluso si algunos nodos no están disponibles**.
- **Modelo de Datos Clave-Valor:** Riak KV sigue un modelo de datos clave-valor. Cada elemento de datos en la base de datos está asociado con una clave única, y los valores pueden ser de cualquier tipo de datos, como cadenas, datos binarios o incluso estructuras complejas como JSON.
- **MapReduce y Búsqueda:** Riak KV admite MapReduce, lo que permite a los usuarios realizar consultas complejas en datos distribuidos. También proporciona opciones de búsqueda para facilitar la recuperación eficiente de datos.
- **API HTTP y Protocol Buffers:** Riak KV ofrece API tanto HTTP como Protocol Buffers para interactuar con la base de datos. Esto lo hace accesible desde una variedad de lenguajes de programación y plataformas.

Riak KV es recomendado para aplicaciones que requieren alta disponibilidad, tolerancia a fallos y escalabilidad, como aquellas en los dominios web, móvil y sistemas distribuidos.

Operaciones de Lectura en Riak

- **Obtener un Valor por Clave:** Para realizar una operación de lectura, se debe proporcionar la clave asociada al valor que se desea recuperar.

La base de datos busca la clave y devuelve el valor correspondiente si existe.

- **Operaciones de Lectura Consistentes:** Se pueden realizar operaciones de lectura con diferentes niveles de consistencia, como "one" (uno), "quorum" o "all" (todos). Estos niveles determinan cuántos nodos deben responder para que la operación se considere exitosa. Por ejemplo, un nivel de consistencia "quorum" significa que la operación será exitosa si al menos la mitad más uno de los nodos responde.
- **Operaciones de Lectura Condicionales:** Se pueden realizar operaciones de lectura condicionales, como "get if not modified" (obtener si no ha sido modificado) o "get if modified" (obtener si ha sido modificado).
- **Recuperación de Múltiples Valores (Batch Get):** Se pueden realizar operaciones de lectura para recuperar múltiples valores en una sola solicitud. Esto es útil para minimizar la latencia al obtener varios valores en una sola operación.
- **Vector Reloj (Vector Clock):** Riak KV utiliza vectores reloj para rastrear la causalidad entre las actualizaciones distribuidas. Los vectores reloj se utilizan para resolver conflictos en operaciones concurrentes y ayudan a determinar el orden causal entre eventos en diferentes nodos.
- **Manejo de Conflictos:** Si hay conflictos debido a operaciones concurrentes en diferentes nodos, Riak KV permite que el usuario implemente estrategias de resolución de conflictos personalizadas.

Operaciones de escritura en Riak

- **Almacenar un Valor con una Clave:** Para realizar una operación de escritura, se proporciona una clave única junto con el valor que se desea almacenar. La base de datos almacena el valor asociado a esa clave.
- **Actualización de un Valor Existente:** Se pueden realizar operaciones de actualización proporcionando la clave y el nuevo valor. La base de datos sobrescribe el valor existente con el nuevo valor.
- **Escrituras Condicionales:** Riak KV admite escrituras condicionales, como "store if not modified" (almacenar si no ha sido modificado) o "store if modified" (almacenar si ha sido modificado). Estas operaciones permiten actualizar un valor sólo si se cumple una condición específica.
- **Vector Reloj (Vector Clock):** Al igual que con las operaciones de lectura, Riak KV utiliza vectores reloj para rastrear la causalidad entre las actualizaciones distribuidas durante las operaciones de escritura. Esto es crucial para resolver conflictos en operaciones concurrentes.
- **Eliminación de Valores:** Se puede eliminar un valor proporcionando la clave asociada.

- **Escrituras en Lotes (Batch):** Al igual que con las lecturas, se pueden realizar operaciones de escritura en lotes, lo que permite realizar múltiples escrituras en una sola solicitud para mejorar la eficiencia.

Indexación en Riak

La indexación en Riak KV se logra mediante índices secundarios, que permiten realizar búsquedas basadas en criterios específicos más allá de la clave primaria. Aquí hay algunas características relacionadas con la indexación en Riak KV:

- **Índices Secundarios:** En Riak KV se admiten índices secundarios que permiten indexar datos en función de atributos específicos. Estos índices secundarios pueden ser creados y gestionados para mejorar el rendimiento de las consultas.
- **Búsqueda de Rangos:** Los índices secundarios en Riak KV admiten búsquedas de rangos, lo que significa que se pueden realizar consultas que recuperan los datos contenidos dentro de un rango específico.
- **Índices Binarios de Conjunto:** Riak KV admite índices binarios de conjunto, lo que permite la indexación de valores múltiples para una clave dada. Esto es útil cuando una clave puede tener múltiples valores asociados.
- **Consultas de Búsqueda Complejas:** Se pueden realizar consultas de búsqueda complejas utilizando índices secundarios combinados con otras operaciones de consulta, como la búsqueda de rangos y la recuperación de múltiples valores.
- **Integración con MapReduce:** Riak KV se integra con el paradigma MapReduce, lo que permite realizar operaciones de búsqueda más complejas y procesamiento de datos a gran escala.

Proyecto en Riak

Para la realización del proyecto en **Riak**, ha sido un poco más complejo que para realizarlo en **MongoDB** ya que **Riak** tiene muy poco soporte y no es tan utilizada como **MongoDB**.

En primer lugar, se ha hecho uso de una máquina virtual con el sistema operativo **Ubuntu 18.04**, ya que es la única versión que he encontrado para poder arrancar **Riak**.

Una vez estamos ya en Ubuntu, es necesario descargarse tanto **Riak (versión 2.2.3-1)** como **VisualStudioCode** (es la herramienta que se ha utilizado para la resolución del proyecto)

Después de descargarlo, para instalarlo, se abrirá una terminal y se ejecutará el siguiente comando:

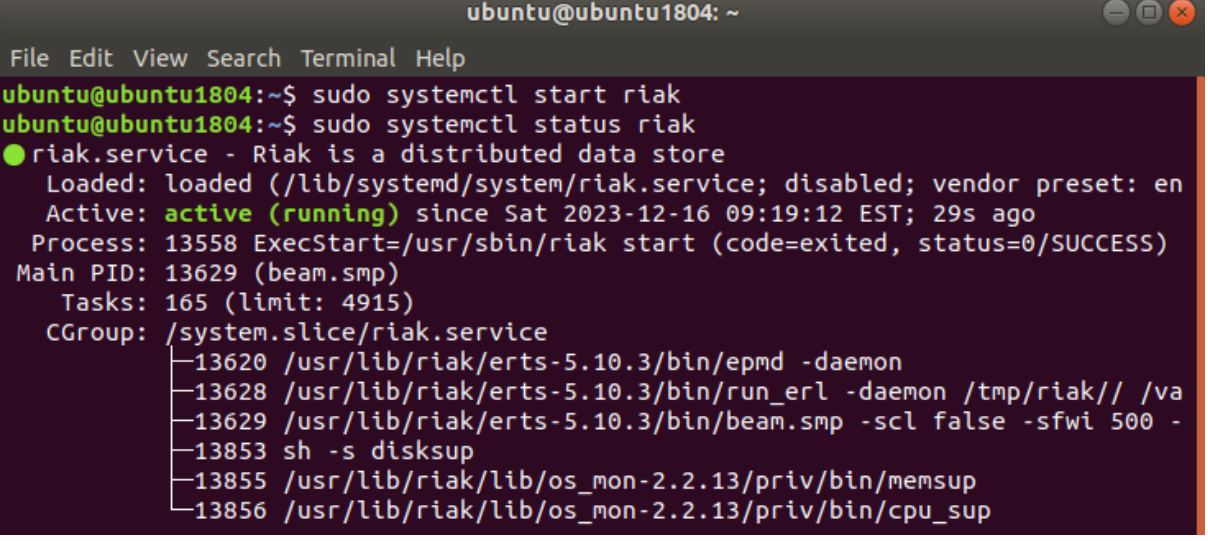
```
sudo dpkg-i-riak-2.3.3-1.amd64.deb
```

Con este comando ya estará instalado riak en nuestro equipo y para ponerlo en funcionamiento se utilizará el siguiente comando:

sudo systemctl start riak

Por último para comprobar el status de **Riak** (comprobar que ha sido iniciado), se utilizará el siguiente comando:

sudo systemctl status riak



```
ubuntu@ubuntu1804: ~
File Edit View Search Terminal Help
ubuntu@ubuntu1804:~$ sudo systemctl start riak
ubuntu@ubuntu1804:~$ sudo systemctl status riak
● riak.service - Riak is a distributed data store
   Loaded: loaded (/lib/systemd/system/riak.service; disabled; vendor preset: en
   Active: active (running) since Sat 2023-12-16 09:19:12 EST; 29s ago
   Process: 13558 ExecStart=/usr/sbin/riak start (code=exited, status=0/SUCCESS)
  Main PID: 13629 (beam.smp)
    Tasks: 165 (limit: 4915)
   CGroup: /system.slice/riak.service
           └─13620 /usr/lib/riak/erts-5.10.3/bin/epmd -daemon
             └─13628 /usr/lib/riak/erts-5.10.3/bin/run_erl -daemon /tmp/riak// /va
               └─13629 /usr/lib/riak/erts-5.10.3/bin/beam.smp -scl false -sfwi 500 -
                 └─13853 sh -s disksup
                   └─13855 /usr/lib/riak/lib/os_mon-2.2.13/priv/bin/memsup
                     └─13856 /usr/lib/riak/lib/os_mon-2.2.13/priv/bin/cpu_sup
```

Como se puede observar en la imagen, **Riak** se ha iniciado correctamente. **Es importante no cerrar esta consola mientras se está ejecutando el proyecto para que no se detenga Riak.**

Para instalar **VisualStudioCode**(versión 1.85), de igual manera que con **Riak**, se descarga desde el navegador y para instalarlo se utilizará el siguiente comando:

sudo dpkg-i-code_1.85.1-1702462158_amd64.deb

Para el proyecto también se ha hecho uso de un entorno virtual, donde poder instalar las distintas **librerías** que se necesitan para el proyecto, de manera que si se ejecuta el proyecto desde otro dispositivo, las **librerías** ya estarán instaladas. Para ello en la terminal del **VisualStudioCode** en la carpeta del proyecto se utilizarán los siguientes comandos (estos comandos son los que se utilizan para Linux, en caso de **Windows** algunos cambian):

python -m venv <NombreDelEntornoVirtual>
source <NombreDelEntornoVirtual>/bin/activate

El primer comando sería para crear el entorno virtual, dándole el nombre que se quiera y el segundo la activa, a partir de este punto cualquier librería que se instale se quedará en el entorno virtual independientemente del equipo en el que se esté ejecutando. Por último para ejecutar el proyecto se utiliza el comando:

python projectPBDriak.py

Este será el código correspondiente al proyecto en Riak:

```
import riak
import datetime
import csv

# Starting Client
myClient = riak.RiakClient(pb_port=8087)

def checkConnection(RiakClient):
    try:
        server_info = RiakClient.ping()

        if server_info:
            print("CONNECTED WITH RIAK", server_info)
            return True
        else:
            print("ERROR, NOT CONNECTED RIAK", server_info)
            return False

    except Exception as e:
        print("Error al intentar conectar con RIAK:", e)

def read_csv():
    with open('Top_rated_movies1.csv', 'r') as file:
        # Read CSV
        csv_reader = csv.reader(file)
        next(csv_reader)

        # Fetch rows
        for row in csv_reader:
            title = str(row[1])
            overview = str(row[2])
            popularity = float(row[3])
            release_date = datetime.datetime.strptime(row[4], "%Y-%m-%d").date().isoformat()
            vote_average = float(row[5])
            vote_count = int(row[6])

            obj = myBucket.new(title, data={'title': title, 'overview': overview, 'popularity': popularity,
                                           'release_date': release_date, 'vote_average': vote_average,
                                           'vote_count': vote_count})

            obj.store()
            # Use date.fromisoformat(date) to show date in good format when getting an object from DB.
            # print('Objeto insertado en RIAK:', obj.key)
```

```

def search_by_year(target_year):
    result_list = []
    my_bucket = myClient.bucket('Top_rated_movies1')

    for key in my_bucket.get_keys():
        obj = my_bucket.get(key)
        release_date_str = obj.data.get('release_date')

        # Convertir la cadena de fecha al formato deseado
        release_date = datetime.datetime.strptime(release_date_str, "%Y-%m-%d").date()

        # Obtener el año de la fecha de lanzamiento
        release_year = release_date.year

        if release_year == target_year:
            title = obj.data.get('title')
            popularity = obj.data.get('popularity')
            vote_average = obj.data.get('vote_average')
            vote_count = obj.data.get('vote_count')
            result_list.append((title, popularity, release_date, vote_average, vote_count))

    print("--> Numero de titulos encontrados:", len(result_list))
    for item in result_list:
        print(item)

    return result_list

def search_by_title(title_query):
    result_list = []
    my_bucket = myClient.bucket('Top_rated_movies1')

    for key in my_bucket.get_keys():
        obj = my_bucket.get(key)
        title = obj.data.get('title').lower()

        if title_query.lower() in title:
            popularity = obj.data.get('popularity')
            release_date = obj.data.get('release_date')
            vote_average = obj.data.get('vote_average')
            vote_count = obj.data.get('vote_count')
            result_list.append((title, popularity, release_date, vote_average, vote_count))

    print("--> Numero de titulos encontrados:", len(result_list))
    for item in result_list:
        print(item)

    return result_list

```

```

def search_by_vote_average(min_vote_average):
    result_list = []
    my_bucket = myClient.bucket('Top_rated_movies1')

    for key in my_bucket.get_keys():
        obj = my_bucket.get(key)
        vote_average = obj.data.get('vote_average')

        if vote_average is not None and vote_average >= min_vote_average:
            title = obj.data.get('title')
            popularity = obj.data.get('popularity')
            release_date = obj.data.get('release_date')
            vote_count = obj.data.get('vote_count')
            result_list.append((title, popularity, release_date, vote_average, vote_count))

    print("--> Numero de titulos encontrados:", len(result_list))
    for item in result_list:
        print(item)

    return result_list

```

```

def search_by_vote_count(min_vote_count):
    result_list = []
    my_bucket = myClient.bucket('Top_rated_movies1')

    for key in my_bucket.get_keys():
        obj = my_bucket.get(key)
        vote_count = obj.data.get('vote_count')

        if vote_count is not None and vote_count >= min_vote_count:
            title = obj.data.get('title')
            popularity = obj.data.get('popularity')
            release_date = obj.data.get('release_date')
            vote_average = obj.data.get('vote_average')
            result_list.append((title, popularity, release_date, vote_average, vote_count))

    print("--> Numero de titulos encontrados:", len(result_list))
    for item in result_list:
        print(item)

    return result_list

```

```

if checkConnection(myClient):
    if myClient.bucket_type('default').bucket('Top_rated_movies1'):
        print("\n")

```



```

print("-----")
print("--> The bucket already exist")
print("--> Deleting all rows in the bucket")
myBucket = myClient.bucket('Top_rated_movies1')
cont = 0
for key in myClient.get_keys(myBucket):
    myBucket.delete(key)
    cont = cont + 1
    #print("Deleted key:", key)
print("--> Deleted: " + str(cont) + " rows correctly")
print("-----")
print("--> Adding new rows to the bucket")
read_csv()
print("--> Added correctly")
print("-----")
print("\n")
while True:
    print("\nMenú de consultas")
    print("1. Consulta por título")
    print("2. Consulta por fecha")
    print("3. Consulta por votos")
    print("4. Consulta por popularidad")
    print("5. Salir")
    opcion = input("Seleccione el tipo de consulta que le gustaría hacer (1-5): ")
    if opcion == "1":
        tit = str(input("Inserte el título: "))
        print("--> Título seleccionado: " + str(tit))
        search_by_title(tit)
    elif opcion == "2":
        year = int(input("Inserte el año: "))
        print("--> Año seleccionado: " + str(year))
        search_by_year(year)
    elif opcion == "3":
        avg = float(input("Inserte el número mínimo de la media de votos: "))
        print("--> Numero seleccionado: " + str(avg))
        search_by_vote_average(avg)
    elif opcion == "4":
        count = int(input("Inserte el número mínimo de popularidad: "))
        print("--> Numero seleccionado: " + str(count))
        search_by_vote_count(count)
    elif opcion == "5":
        print("Saliendo del progrma")
        break
    else:
        print("La opción que ha seleccionado no es válida")

else:
    print("--> Creating new bucket")

```

```
# Creating Buckets
myBucket = myClient.bucket('Top_rated_movies1')
print("--> Adding new rows to the bucket")
read_csv()
print("--> Added correctly")
```

Cada vez que se ejecuta, se borran todos los datos que haya del cubo para volverlo a insertar. Como son pocos datos, no tarda mucho en insertarlos, pero en caso de que fueran grandes volúmenes de datos sería recomendable controlarlo para que una vez estén insertados no haga falta borrarlos e insertarlos de nuevo.

A diferencia de **MongoDB**, para comprobar los datos que se han insertado en la **Base de Datos** no existe ninguna interfaz gráfica, por lo tanto, sería necesario realizar una consulta para poder ver el contenido del **cubo (bucket)**. A la hora de mostrar el resultado obtenido por la consulta, no se muestra el campo “**Overview**” ya que ensucia demasiado la salida.

- En primer lugar, se hará una consulta por el título de la película, esta consulta se ha implementado de manera que no sea necesario escribir el nombre completo entero, sino que exista la cadena que se ha introducido por consola en el título de la película. Para ello se ha implementado el siguiente método:

```
def search_by_title(title_query):
    result_list = []
    my_bucket = myClient.bucket('Top_rated_movies1')

    for key in my_bucket.get_keys():
        obj = my_bucket.get(key)
        title = obj.data.get('title').lower()

        if title_query.lower() in title:
            popularity = obj.data.get('popularity')
            release_date = obj.data.get('release_date')
            vote_average = obj.data.get('vote_average')
            vote_count = obj.data.get('vote_count')
            result_list.append((title, popularity, release_date, vote_average, vote_count))

    print("--> Numero de titulos encontrados:", len(result_list))
    for item in result_list:
        print(item)

    return result_list
```

La salida por consola sería la siguiente:

Menú de consultas

1. Consulta por título
2. Consulta por fecha
3. Consulta por votos

4. Consulta por popularidad

5. Salir

Seleccione el tipo de consulta que le gustaría hacer (1-5): 1

Inserte el título: nado

--> Título seleccionado: nado

--> Numero de titulos encontrados: 5

('sharknado 4: the 4th awakens', 17.537, '2016-07-31', 4.6, 354)

('sharknado 2: the second one', 19.079, '2014-07-31', 4.584, 611)

('sharknado', 13.862, '2013-07-11', 3.917, 1482)

('sharknado 5: global swarming', 16.51, '2017-08-06', 4.779, 303)

('sharknado 3: oh hell no!', 17.707, '2015-07-22', 4.635, 466)

Como se puede observar, devuelve todas las tuplas que tengan la cadena “**nado**” en su título.

- En segundo lugar una consulta por una fecha en concreto, donde el usuario pueda buscar películas que se hayan estrenado en un año en concreto. De igual manera que el método anterior se ha hecho uso de una expresión regular para poder utilizar solo el año. Para ello se ha implementado el siguiente método:

def search_by_year(target_year):

 result_list = []

 my_bucket = myClient.bucket('Top Rated Movies1')

 for key in my_bucket.get_keys():

 obj = my_bucket.get(key)

 release_date_str = obj.data.get('release_date')

 release_date = datetime.datetime.strptime(release_date_str, "%Y-%m-%d").date()

 release_year = release_date.year

 if release_year == target_year:

 title = obj.data.get('title')

 popularity = obj.data.get('popularity')

 vote_average = obj.data.get('vote_average')

 vote_count = obj.data.get('vote_count')

 result_list.append((title, popularity, release_date, vote_average, vote_count))

 print("--> Numero de titulos encontrados:", len(result_list))

 for item in result_list:

 print(item)

 return result_list

La salida por consola sería la siguiente:

Menú de consultas

1. Consulta por título

2. Consulta por fecha

3. Consulta por votos

4. Consulta por popularidad

5. Salir

Seleccione el tipo de consulta que le gustaría hacer (1-5): 2

Inserte el año: 2023

--> Año seleccionado: 2023

--> Numero de titulos encontrados: 114

('Carl's Date', 206.128, datetime.date(2023, 6, 15), 7.624, 310)
('Fast X', 749.903, datetime.date(2023, 5, 17), 7.208, 4335)
('Through My Window: Across the Sea', 67.71, datetime.date(2023, 6, 23), 6.547, 591)
('Guy Ritchie's The Covenant', 161.045, datetime.date(2023, 4, 19), 7.8, 1845)
('The Equalizer 3', 573.141, datetime.date(2023, 8, 30), 7.42, 1707)
('Insidious: The Red Door', 132.784, datetime.date(2023, 7, 5), 6.761, 1328)
('Totally Killer', 94.283, datetime.date(2023, 9, 28), 6.947, 487)
('Infinity Pool', 55.225, datetime.date(2023, 1, 27), 6.152, 474)
('Murder Mystery 2', 83.176, datetime.date(2023, 3, 28), 6.503, 1412)
('John Wick: Chapter 4', 383.995, datetime.date(2023, 3, 22), 7.792, 5028)
('Saw X', 509.974, datetime.date(2023, 9, 26), 7.413, 1150)
('Kandahar', 168.575, datetime.date(2023, 5, 25), 6.929, 729)
('Plane', 102.687, datetime.date(2023, 1, 12), 6.99, 1857)
('The Super Mario Bros. Movie', 502.499, datetime.date(2023, 4, 5), 7.75, 7233)
('Boston Strangler', 38.693, datetime.date(2023, 3, 17), 6.73, 510)
('Guardians of the Galaxy Vol. 3', 278.052, datetime.date(2023, 5, 3), 8.008, 5356)
('A Haunting in Venice', 168.804, datetime.date(2023, 9, 13), 6.781, 1244)
('The Last Voyage of the Demeter', 191.678, datetime.date(2023, 8, 9), 7.154, 939)
('Hidden Strike', 112.597, datetime.date(2023, 7, 6), 7.029, 798)
('Knights of the Zodiac', 185.15, datetime.date(2023, 4, 27), 6.6, 894)
('Beautiful Disaster', 49.719, datetime.date(2023, 4, 4), 6.601, 503)
('Haunted Mansion', 86.685, datetime.date(2023, 7, 26), 6.608, 727)
('Sound of Freedom', 415.938, datetime.date(2023, 7, 3), 8.085, 1512)
('The Strays', 33.593, datetime.date(2023, 2, 17), 5.492, 308)
('Elemental', 491.164, datetime.date(2023, 6, 14), 7.725, 3053)
('No Hard Feelings', 174.475, datetime.date(2023, 6, 15), 7.104, 1790)
('Talk to Me', 338.907, datetime.date(2023, 7, 26), 7.195, 1806)
('The Exorcist: Believer', 376.108, datetime.date(2023, 10, 4), 6.165, 601)
('Five Nights at Freddy's', 1252.998, datetime.date(2023, 10, 25), 7.888, 2516)
('The Creator', 1622.502, datetime.date(2023, 9, 27), 7.149, 1114)
('The Wonderful Story of Henry Sugar', 44.038, datetime.date(2023, 9, 20), 7.3, 494)
('Beau Is Afraid', 40.552, datetime.date(2023, 4, 14), 6.807, 618)
('Mission: Impossible - Dead Reckoning Part One', 762.172, datetime.date(2023, 7, 8), 7.586, 2534)
('The Flash', 336.782, datetime.date(2023, 6, 13), 6.837, 3262)
('Anatomy of a Fall', 44.767, datetime.date(2023, 8, 23), 7.819, 345)
('Cocaine Bear', 93.341, datetime.date(2023, 2, 22), 6.2, 1595)
('Kill Boksoon', 48.035, datetime.date(2023, 2, 17), 6.948, 329)
('Operation Fortune: Ruse de Guerre', 77.61, datetime.date(2023, 1, 4), 6.57, 1107)
('Evil Dead Rise', 133.0, datetime.date(2023, 4, 12), 6.972, 2467)

('Asterix & Obelix: The Middle Kingdom', 48.019, datetime.date(2023, 2, 1), 4.901, 754)
 ('Darkland: The Return', 69.678, datetime.date(2023, 4, 13), 6.62, 304)
 ('Knock at the Cabin', 66.77, datetime.date(2023, 2, 1), 6.333, 1990)
 ('Peter Pan & Wendy', 77.869, datetime.date(2023, 4, 20), 5.66, 559)
 ('Ballerina', 147.038, datetime.date(2023, 10, 5), 6.979, 310)
 ('Sharper', 35.052, datetime.date(2023, 2, 10), 7.063, 366)
 ('Renfield', 65.153, datetime.date(2023, 4, 7), 6.698, 1144)
 ('Dungeons & Dragons: Honor Among Thieves', 143.276, datetime.date(2023, 3, 23), 7.421, 2631)
 ('Tin & Tina', 28.291, datetime.date(2023, 3, 24), 6.011, 321)
 ('Tetris', 43.082, datetime.date(2023, 3, 15), 7.78, 945)
 ('Barbie', 484.841, datetime.date(2023, 7, 19), 7.192, 6017)
 ('Ghosted', 145.022, datetime.date(2023, 4, 18), 7.035, 1401)
 ('Extraction 2', 138.294, datetime.date(2023, 6, 9), 7.477, 1946)
 ('Magic Mike's Last Dance', 39.259, datetime.date(2023, 2, 9), 6.748, 379)
 ('Nimona', 66.623, datetime.date(2023, 6, 23), 7.972, 596)
 ('After Everything', 284.739, datetime.date(2023, 9, 13), 7.037, 551)
 ('All Your Faces', 21.525, datetime.date(2023, 3, 29), 8.107, 304)
 ('The Devil Conspiracy', 85.265, datetime.date(2023, 1, 13), 6.489, 316)
 ('Reptile', 54.225, datetime.date(2023, 9, 29), 6.85, 511)
 ('JUNG_E', 54.719, datetime.date(2023, 1, 12), 6.218, 518)
 ('We Have a Ghost', 46.28, datetime.date(2023, 2, 24), 6.534, 471)
 ('Srimulat: Hidup Mengang Komedi', 11.038, datetime.date(2023, 11, 23), 4.0, 1)
 ('Gran Turismo', 481.406, datetime.date(2023, 8, 9), 7.992, 1504)
 ('Luther: The Fallen Sun', 35.755, datetime.date(2023, 2, 24), 6.735, 755)
 ('Winnie the Pooh: Blood and Honey', 68.676, datetime.date(2023, 1, 27), 5.281, 1038)
 ('The Mother', 79.358, datetime.date(2023, 5, 4), 6.746, 1041)
 ('The Boogeyman', 70.987, datetime.date(2023, 5, 31), 6.5, 628)
 ('The Three Musketeers: D'Artagnan', 56.615, datetime.date(2023, 4, 5), 7.133, 756)
 ('Nowhere', 256.473, datetime.date(2023, 9, 29), 7.537, 880)
 ('The Black Demon', 62.619, datetime.date(2023, 4, 26), 6.191, 551)
 ('Heart of Stone', 182.914, datetime.date(2023, 8, 9), 6.881, 1333)
 ('The Killer', 258.899, datetime.date(2023, 10, 25), 6.652, 943)
 ('They Cloned Tyrone', 33.915, datetime.date(2023, 6, 14), 6.716, 402)
 ('Your Place or Mine', 31.875, datetime.date(2023, 2, 10), 6.309, 623)
 ('Red, White & Royal Blue', 103.594, datetime.date(2023, 7, 27), 8.1, 899)
 ('Expend4bles', 873.341, datetime.date(2023, 9, 15), 6.438, 792)
 ('Mummies', 71.109, datetime.date(2023, 1, 5), 7.116, 392)
 ('To Catch a Killer', 54.831, datetime.date(2023, 4, 6), 7.0, 601)
 ('Mixed by Erry', 18.847, datetime.date(2023, 3, 2), 7.484, 334)
 ('Resident Evil: Death Island', 133.656, datetime.date(2023, 6, 22), 7.513, 747)
 ('The Nun II', 419.542, datetime.date(2023, 9, 6), 6.915, 1460)
 ('Transformers: Rise of the Beasts', 460.082, datetime.date(2023, 6, 6), 7.458, 3588)
 ('Cobweb', 211.249, datetime.date(2023, 7, 19), 6.792, 477)
 ('Oppenheimer', 1766.305, datetime.date(2023, 7, 19), 8.163, 4905)
 ('AKA', 47.062, datetime.date(2023, 4, 28), 6.903, 542)
 ('Indiana Jones and the Dial of Destiny', 260.195, datetime.date(2023, 6, 28), 6.678, 2081)
 ('Shazam! Fury of the Gods', 165.951, datetime.date(2023, 3, 15), 6.645, 2543)

('My Fault', 310.835, datetime.date(2023, 6, 8), 8.098, 1944)
 ('Love at First Sight', 67.616, datetime.date(2023, 9, 15), 7.304, 334)
 ('Mavka: The Forest Song', 298.215, datetime.date(2023, 3, 2), 7.285, 509)
 ('Zom 100: Bucket List of the Dead', 60.085, datetime.date(2023, 8, 3), 6.795, 439)
 ('Ruby Gillman, Teenage Kraken', 182.232, datetime.date(2023, 6, 28), 7.396, 758)
 ('Sisu', 143.412, datetime.date(2023, 1, 27), 7.535, 1502)
 ('Creed III', 163.734, datetime.date(2023, 3, 1), 7.182, 2089)
 ('Teenage Mutant Ninja Turtles: Mutant Mayhem', 307.152, datetime.date(2023, 7, 31), 7.312, 851)
 ('Ant-Man and the Wasp: Quantumania', 192.863, datetime.date(2023, 2, 15), 6.395, 4215)
 ('The Hunger Games: The Ballad of Songbirds & Snakes', 595.362, datetime.date(2023, 11, 15), 7.2, 401)
 ('Straits', 140.092, datetime.date(2023, 8, 17), 7.55, 527)
 ('Die Hart', 49.077, datetime.date(2023, 2, 22), 6.0, 350)
 ('Scream VI', 183.313, datetime.date(2023, 3, 8), 7.136, 2065)
 ('65', 114.682, datetime.date(2023, 3, 2), 6.109, 1830)
 ('Flamin' Hot', 103.072, datetime.date(2023, 3, 11), 8.236, 432)
 ('Paradise', 31.999, datetime.date(2023, 6, 24), 6.686, 411)
 ('Meg 2: The Trench', 503.411, datetime.date(2023, 8, 2), 6.745, 2586)
 ('The Pope's Exorcist', 132.821, datetime.date(2023, 4, 5), 7.103, 2197)
 ('The Wandering Earth II', 149.717, datetime.date(2023, 1, 22), 7.234, 396)
 ('No One Will Save You', 119.501, datetime.date(2023, 9, 22), 6.807, 753)
 ('You People', 42.969, datetime.date(2023, 1, 20), 5.743, 586)
 ('The Last Kingdom: Seven Kings Must Die', 81.714, datetime.date(2023, 4, 14), 7.295, 550)
 ('Teen Wolf: The Movie', 50.302, datetime.date(2023, 1, 18), 7.631, 695)
 ('The Marvels', 480.065, datetime.date(2023, 11, 8), 6.571, 539)
 ('Asteroid City', 58.538, datetime.date(2023, 6, 8), 6.581, 1171)
 ('Spider-Man: Across the Spider-Verse', 448.515, datetime.date(2023, 5, 31), 8.403, 4976)
 ('Blue Beetle', 515.987, datetime.date(2023, 8, 16), 6.9, 1714)
 ('Miraculous: Ladybug & Cat Noir, The Movie', 204.454, datetime.date(2023, 7, 5), 7.8, 663)

Como se puede observar en la salida, devuelve todas las películas que han sido lanzadas en **2023**.

- En tercer lugar una consulta por número mínimo de la media de votos de una película, donde el usuario introducirá un valor que podrá ser decimal para poner un límite mínimo de media de valoración de una película. Para ello se ha implementado el siguiente método:

```
def search_by_vote_average(min_vote_average):
```

```
    result_list = []
```

```
    my_bucket = myClient.bucket('Top_rated_movies1')
```

```
    for key in my_bucket.get_keys():
```

```
        obj = my_bucket.get(key)
```

```
        vote_average = obj.data.get('vote_average')
```

```
        if vote_average is not None and vote_average >= min_vote_average:
```

```

        title = obj.data.get('title')
        popularity = obj.data.get('popularity')
        release_date = obj.data.get('release_date')
        vote_count = obj.data.get('vote_count')
        result_list.append((title, popularity, release_date, vote_average, vote_count))

    print("--> Numero de titulos encontrados:", len(result_list))
    for item in result_list:
        print(item)

    return result_list

```

La salida por consola sería la siguiente:

Menú de consultas

1. Consulta por título

2. Consulta por fecha

3. Consulta por votos

4. Consulta por popularidad

5. Salir

Seleccione el tipo de consulta que le gustaría hacer (1-5): 3

Inserte el número mínimo de la media de votos: 8.5

--> Numero seleccionado: 8.5

--> Numero de titulos encontrados: 11

```

('12 Angry Men', 66.507, '1957-04-10', 8.545, 7805)
('The Green Mile', 100.097, '1999-12-10', 8.509, 16159)
('Parasite', 106.749, '2019-05-30', 8.515, 16684)
('The Shawshank Redemption', 179.959, '1994-09-23', 8.705, 24985)
('Dilwale Dulhania Le Jayenge', 43.081, '1995-10-20', 8.546, 4282)
('Spirited Away', 104.619, '2001-07-20', 8.54, 15131)
('Your Name.', 112.597, '2016-08-26', 8.505, 10513)
('The Godfather Part II', 106.688, '1974-12-20', 8.589, 11467)
('The Dark Knight', 127.08, '2008-07-16', 8.514, 30958)
('Schindler's List', 81.656, '1993-12-15', 8.571, 14813)
('The Godfather', 167.536, '1972-03-14', 8.708, 18991)

```

Como se puede observar en la salida, se devuelven todas las películas que tengan mínimo un **8.5** de media de votos.

- Para la cuarta y última consulta, se trata de una consulta por popularidad, de manera que el usuario podrá buscar una película que supere un mínimo de popularidad. Para ello se ha implementando el siguiente método:

```

def search_by_vote_count(min_vote_count):

```

```

    result_list = []

```

```

    my_bucket = myClient.bucket('Top-rated-movies1')

```

```

    for key in my_bucket.get_keys():

```

```

obj = my_bucket.get(key)
vote_count = obj.data.get('vote_count')

if vote_count is not None and vote_count >= min_vote_count:
    title = obj.data.get('title')
    popularity = obj.data.get('popularity')
    release_date = obj.data.get('release_date')
    vote_average = obj.data.get('vote_average')
    result_list.append((title, popularity, release_date, vote_average, vote_count))

print("--> Numero de titulos encontrados:", len(result_list))
for item in result_list:
    print(item)

return result_list

```

La salida por consola sería la siguiente:

Menú de consultas

1. Consulta por título

2. Consulta por fecha

3. Consulta por votos

4. Consulta por popularidad

5. Salir

Seleccione el tipo de consulta que le gustaría hacer (1-5): 4

Inserte el número mínimo de popularidad: 16000

--> Numero seleccionado: 16000

--> Number of titles found: 91

('Pirates of the Caribbean: The Curse of the Black Pearl', 124.301, '2003-07-09', 7.796, 19420)

('Spider-Man', 104.022, '2002-05-01', 7.283, 17828)

('The Shawshank Redemption', 179.959, '1994-09-23', 8.705, 24985)

('Kill Bill: Vol. 1', 62.354, '2003-10-10', 7.971, 16340)

('The Shining', 51.925, '1980-05-23', 8.219, 16396)

('The Incredibles', 96.659, '2004-10-27', 7.705, 16736)

('The Truman Show', 77.672, '1998-06-04', 8.135, 17052)

('Avengers: Age of Ultron', 119.33, '2015-04-22', 7.274, 21924)

('Avengers: Infinity War', 304.697, '2018-04-25', 8.252, 28003)

('Finding Nemo', 95.279, '2003-05-30', 7.825, 18221)

('The Lord of the Rings: The Return of the King', 129.449, '2003-12-01', 8.476, 22601)

('Harry Potter and the Order of the Phoenix', 133.528, '2007-07-08', 7.685, 18342)

('Thor', 101.147, '2011-04-21', 6.766, 20161)

('The Wolf of Wall Street', 103.42, '2013-12-25', 8.034, 22488)

('Gladiator', 82.685, '2000-05-04', 8.21, 17183)

('Harry Potter and the Chamber of Secrets', 160.38, '2002-11-13', 7.718, 20602)

('The Intouchables', 70.159, '2011-11-02', 8.28, 16274)

('Pulp Fiction', 86.057, '1994-09-10', 8.488, 26185)

('Inside Out', 251.229, '2015-06-09', 7.922, 19688)
 ('Iron Man', 120.877, '2008-04-30', 7.639, 25076)
 ('Monsters, Inc.', 141.907, '2001-11-01', 7.837, 17364)
 ('Thor: The Dark World', 95.18, '2013-10-30', 6.533, 16487)
 ('Shutter Island', 76.883, '2010-02-14', 8.201, 22551)
 ('Spider-Man: Homecoming', 122.603, '2017-07-05', 7.344, 20666)
 ('The Amazing Spider-Man', 84.722, '2012-06-23', 6.693, 16479)
 ('Captain America: The First Avenger', 66.911, '2011-07-22', 6.995, 20413)
 ('Bohemian Rhapsody', 67.648, '2018-10-24', 7.995, 16075)
 ('Gone Girl', 109.965, '2014-10-01', 7.892, 17550)
 ('The Lord of the Rings: The Fellowship of the Ring', 124.683, '2001-12-18', 8.405, 23569)
 ('Fantastic Beasts and Where to Find Them', 91.549, '2016-11-16', 7.332, 18005)
 ('Star Wars: The Force Awakens', 64.136, '2015-12-15', 7.289, 18487)
 ('Captain America: Civil War', 121.118, '2016-04-27', 7.442, 21694)
 ('Thor: Ragnarok', 120.878, '2017-10-02', 7.595, 19691)
 ('Coco', 190.422, '2017-10-27', 8.218, 18199)
 ('The Lord of the Rings: The Two Towers', 108.444, '2002-12-18', 8.387, 20490)
 ('John Wick', 75.559, '2014-10-22', 7.423, 18108)
 ('The Dark Knight Rises', 90.781, '2012-07-17', 7.777, 21539)
 ('The Matrix', 94.122, '1999-03-30', 8.209, 24108)
 ('Avatar', 151.812, '2009-12-15', 7.575, 30044)
 ('Get Out', 68.487, '2017-02-24', 7.619, 16276)
 ('Harry Potter and the Half-Blood Prince', 137.319, '2009-07-15', 7.695, 18292)
 ('Batman v Superman: Dawn of Justice', 86.15, '2016-03-23', 5.956, 17203)
 ('Logan', 88.509, '2017-02-28', 7.818, 18249)
 ('Avengers: Endgame', 166.118, '2019-04-24', 8.3, 24104)
 ('Split', 76.881, '2017-01-19', 7.338, 16452)
 ('The Green Mile', 100.097, '1999-12-10', 8.509, 16159)
 ('WALL·E', 101.062, '2008-06-22', 8.081, 17609)
 ('Black Panther', 86.107, '2018-02-13', 7.388, 21197)
 ('Spider-Man: No Way Home', 383.751, '2021-12-15', 7.986, 18560)
 ('Deadpool', 110.954, '2016-02-09', 7.607, 29102)
 ('Star Wars', 104.667, '1977-05-25', 8.204, 19340)
 ('Captain America: The Winter Soldier', 57.712, '2014-03-20', 7.669, 17834)
 ('Forrest Gump', 132.688, '1994-06-23', 8.476, 25699)
 ('Parasite', 106.749, '2019-05-30', 8.515, 16684)
 ('The Imitation Game', 65.09, '2014-11-14', 8.007, 16110)
 ('The Hunger Games: Catching Fire', 196.18, '2013-11-15', 7.422, 16360)
 ('Guardians of the Galaxy', 54.527, '2014-07-30', 7.906, 26834)
 ('Harry Potter and the Philosopher's Stone', 191.153, '2001-11-16', 7.915, 25646)
 ('Ant-Man', 80.476, '2015-07-14', 7.081, 18856)
 ('Harry Potter and the Deathly Hallows: Part 2', 154.583, '2011-07-12', 8.103, 19370)
 ('Deadpool 2', 120.647, '2018-05-10', 7.491, 16475)
 ('It', 80.336, '2017-09-06', 7.243, 18184)
 ('Titanic', 132.667, '1997-11-18', 7.9, 23876)
 ('The Revenant', 77.144, '2015-12-25', 7.528, 17249)
 ('Interstellar', 170.639, '2014-11-05', 8.42, 32975)
 ('Mad Max: Fury Road', 153.395, '2015-05-13', 7.585, 21073)

('Back to the Future', 76.112, '1985-07-03', 8.313, 18672)
 ('Batman Begins', 73.493, '2005-06-10', 7.703, 19774)
 ('The Hobbit: An Unexpected Journey', 92.223, '2012-12-12', 7.342, 17496)
 ('The Godfather', 167.536, '1972-03-14', 8.708, 18991)
 ('The Hunger Games', 131.184, '2012-03-12', 7.198, 20851)
 ('Harry Potter and the Goblet of Fire', 161.512, '2005-11-16', 7.815, 19473)
 ('Arrival', 78.86, '2016-11-10', 7.591, 16700)
 ('Harry Potter and the Deathly Hallows: Part 1', 147.492, '2010-11-17', 7.761, 17947)
 ('Joker', 103.754, '2019-10-01', 8.165, 23683)
 ('Harry Potter and the Prisoner of Azkaban', 204.214, '2004-05-31', 8.019, 20242)
 ('The Martian', 75.54, '2015-09-30', 7.682, 18673)
 ('Toy Story', 123.853, '1995-10-30', 7.971, 17326)
 ('Iron Man 3', 117.109, '2013-04-18', 6.926, 21212)
 ('Inglourious Basterds', 92.913, '2009-08-19', 8.216, 20982)
 ('Doctor Strange', 98.319, '2016-10-25', 7.427, 21080)
 ('Up', 123.427, '2009-05-28', 7.952, 19051)
 ('The Maze Runner', 173.001, '2014-09-10', 7.174, 16034)
 ('The Dark Knight', 127.08, '2008-07-16', 8.514, 30958)
 ('Inception', 154.283, '2010-07-15', 8.365, 34794)
 ('Se7en', 74.831, '1995-09-22', 8.37, 19651)
 ('Django Unchained', 72.629, '2012-12-25', 8.173, 24899)
 ('Fight Club', 105.447, '1999-10-15', 8.44, 27556)
 ('Jurassic World', 70.28, '2015-06-06', 6.683, 19541)
 ('Suicide Squad', 58.123, '2016-08-03', 5.91, 20240)
 ('Iron Man 2', 146.838, '2010-04-28', 6.834, 19907)

En la salida se pueden ver todos los títulos que superan el valor **16000** de popularidad.

Bibliografía

Documentación MongoDB: <https://www.mongodb.com/docs/manual/crud/>

Documentación expresiones regulares python: <https://docs.python.org/es/3/howto/regex.html>

Documentación de bases de datos NoSQL y tipos: <https://www.ibm.com/es-es/topics/nosql-databases>

Descarga de la maquina virtual ubuntu 18.04: <https://www.linuxvmimages.com/images/ubuntu-1804/>

Documentación de Riak: <https://docs.riak.com/riak/kv/latest/index.html>

Descargar herramienta VisualStudioCode: <https://code.visualstudio.com/download>