EVALUACIÓN 3 - Trabajo con MongoDB

Curso: DPDC-105-BIG DATA:S1 (2022)

Diplomado en Data Science

Fecha de entrega: Sábado 06 de Agosto2022

Grupo 3:

- Nombre integrante 1: Juan Pablo González Collao
- Nombre integrante 2: Pablo Omar Walters Barraza

Documentación oficial: https://www.mongodb.com/docs/.

Como utilizar un Jupyter notebook en un entorno virutal

Sitio : https://janakiev.com/blog/jupyter-virtual-envs/

Intrucciones

```
python3 -m venv env
pip install --user ipykernel
python -m ipykernel install --user --name=env
```

Instalar Mongo DB en OSX (Utilizando brew)

```
brew tap mongodb/brew
brew install mongodb-community@5.0
brew services start mongodb-community@5.0
brew services stop mongodb-community@5.0
```

In [1]:
```
# Instalación de bibliotecas necesarias
!pip3 install pandas
!pip3 install pymongo
```

```
Requirement already satisfied: pandas in c:\users\milan\anaconda3\lib\site-packages (1.4.2)
Requirement already satisfied: numpy>=1.18.5 in c:\users\milan\anaconda3\lib\site-packages (from pandas) (1.21.
5)
Requirement already satisfied: pytz>=2020.1 in c:\users\milan\anaconda3\lib\site-packages (from pandas) (2022.
1)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\milan\anaconda3\lib\site-packages (from panda
s) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\milan\anaconda3\lib\site-packages (from python-dateutil>=2.
8.1->pandas) (1.16.0)
Requirement already satisfied: pymongo in c:\users\milan\anaconda3\lib\site-packages (3.12.0)
```

In [2]:
```
import pandas as pd
import json
```

In [3]:
```
# Lectura de archivos CSV
df_films        = pd.read_csv('film.csv')
df_films_actor  = pd.read_csv('film_actor.csv')
df_actor        = pd.read_csv('actor.csv')
film_cate= pd.read_csv('film_category.csv')
cate= pd.read_csv('category.csv')
film_text= pd.read_csv('film_text.csv')
```

In [4]:
```
film_text
```

Out[4]:

	film_id	title	description
0	1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist...
1	2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrat...
2	3	ADAPTATION HOLES	A Astounding Reflection of a Lumberjack And a ...
3	4	AFFAIR PREJUDICE	A Fanciful Documentary of a Frisbee And a Lumb...
4	5	AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef And ...
...	...	...	...
995	996	YOUNG LANGUAGE	A Unbelieveable Yarn of a Boat And a Database ...
996	997	YOUTH KICK	A Touching Drama of a Teacher And a Cat who mu...
997	998	ZHIVAGO CORE	A Fateful Yarn of a Composer And a Man who mus...
998	999	ZOOLANDER FICTION	A Fateful Reflection of a Waitress And a Boat ...
999	1000	ZORRO ARK	A Intrepid Panorama of a Mad Scientist And a B...

1000 rows × 3 columns

In [5]:
```
film_text
desc= pd.DataFrame(film_text.groupby('film_id').apply(lambda x: x.to_json(orient='records')), columns=['descript
desc['description']=desc['description'].apply(lambda x: json.loads(x))
desc
```

Out[5]:

	film_id	description
0	1	[{'film_id': 1, 'title': 'ACADEMY DINOSAUR', '...
1	2	[{'film_id': 2, 'title': 'ACE GOLDFINGER', 'de...
2	3	[{'film_id': 3, 'title': 'ADAPTATION HOLES', '...
3	4	[{'film_id': 4, 'title': 'AFFAIR PREJUDICE', '...
4	5	[{'film_id': 5, 'title': 'AFRICAN EGG', 'descr...
...	...	...
995	996	[{'film_id': 996, 'title': 'YOUNG LANGUAGE', '...
996	997	[{'film_id': 997, 'title': 'YOUTH KICK', 'desc...
997	998	[{'film_id': 998, 'title': 'ZHIVAGO CORE', 'de...
998	999	[{'film_id': 999, 'title': 'ZOOLANDER FICTION'...
999	1000	[{'film_id': 1000, 'title': 'ZORRO ARK', 'desc...

1000 rows × 2 columns

1. Actualizar la Colección en la BD

Agregue la categoría de las películas ocupando Python y Pandas a la colección definida en el Notebook y actualice la colección en la base de datos.

Se unen las dos variables a través del id de categoría, para agrear la categoría a cada filM, eliminando los last_update.

In [6]:
```
categoria = pd.merge(film_cate, cate, how='inner', on='category_id').drop(['last_update_x','last_update_y'], ax
categoria
```

Out[6]:

	film_id	category_id	name
0	1	6	Documentary
1	3	6	Documentary
2	40	6	Documentary
3	58	6	Documentary
4	62	6	Documentary
...	...	...	...
995	928	3	Children
996	955	3	Children
997	959	3	Children
998	993	3	Children
999	999	3	Children

1000 rows × 3 columns

Se agrupa las categorías en base al id de film, haciendo un arreglo. se transforma a Json.

In [7]:
```
aggr_categoria = pd.DataFrame(categoria.groupby('film_id').apply(lambda x: x.to_json(orient='records')), columns
aggr_categoria['category'] = aggr_categoria['category'].apply(lambda x: json.loads(x))
aggr_categoria
```

Out[7]:

	film_id	category
0	1	[{'film_id': 1, 'category_id': 6, 'name': 'Doc...
1	2	[{'film_id': 2, 'category_id': 11, 'name': 'Ho...
2	3	[{'film_id': 3, 'category_id': 6, 'name': 'Doc...
3	4	[{'film_id': 4, 'category_id': 11, 'name': 'Ho...
4	5	[{'film_id': 5, 'category_id': 8, 'name': 'Fam...
...	...	...
995	996	[{'film_id': 996, 'category_id': 6, 'name': 'D...
996	997	[{'film_id': 997, 'category_id': 12, 'name': '...
997	998	[{'film_id': 998, 'category_id': 11, 'name': '...
998	999	[{'film_id': 999, 'category_id': 3, 'name': 'C...
999	1000	[{'film_id': 1000, 'category_id': 5, 'name': '...

1000 rows × 2 columns

In [8]:
```
# Mostramos un sample de actor y films
```

In [9]:
```
df_films_actor.sample()
```

Out[9]:

	actor_id	film_id	last_update
893	35	256	2006-02-15 05:05:03

In [10]:
```
# Agregamos los actores a la relación N:N (film, actor)
df_films_actor_agg = pd.merge(df_films_actor,
                              df_actor,
                              how='left',
                              on='actor_id')


df_films_actor_agg.head()
```

Out[10]:

	actor_id	film_id	last_update_x	first_name	last_name	last_update_y
0	1	1	2006-02-15 05:05:03	PENELOPE	GUINESS	2006-02-15 04:34:33
1	1	23	2006-02-15 05:05:03	PENELOPE	GUINESS	2006-02-15 04:34:33
2	1	25	2006-02-15 05:05:03	PENELOPE	GUINESS	2006-02-15 04:34:33
3	1	106	2006-02-15 05:05:03	PENELOPE	GUINESS	2006-02-15 04:34:33
4	1	140	2006-02-15 05:05:03	PENELOPE	GUINESS	2006-02-15 04:34:33

In [11]:
```
# Agrupo por film todos los actores y lo transformo a un archivo json.

df_films_actor_agg_json = df_films_actor_agg.groupby('film_id').apply(lambda x: x.to_json(orient='records'))


df_films_actor_agg_json = pd.DataFrame(df_films_actor_agg_json).reset_index().rename({0:'actors'}, axis='columns

df_films_actor_agg_json['actors'] = df_films_actor_agg_json['actors'].apply(lambda x: json.loads(x))

#df_films_actor_agg_json['actors'][0]
```

In [12]:
```
# Finalmente agregamos nuestra lista de json a films
df_new_films = pd.merge(df_films,
                        df_films_actor_agg_json,
                        how='left',
                        on='film_id')

df_new_films= pd.merge(df_new_films,
                               aggr_categoria,
                               how='left',
                               on='film_id')
#df_new_films= pd.merge(df_new_films,desc,how='left', on='film_id')


df_new_films.head()
```

Out[12]:

	film_id	title	description	release_year	language_id	original_language_id	rental_duration	rental_rate	length	replacen
0	1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist...	2006	1	NaN	6	0.99	86	
1	2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrat...	2006	1	NaN	3	4.99	48	
2	3	ADAPTATION HOLES	A Astounding Reflection of a Lumberjack And a ...	2006	1	NaN	7	2.99	50	
3	4	AFFAIR PREJUDICE	A Fanciful Documentary of a Frisbee And a Lumb...	2006	1	NaN	5	2.99	117	
4	5	AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef And ...	2006	1	NaN	6	2.99	130	

In [13]:
```
# Ejemplo de un documento que vamos a subir a mongo DB
df_new_films.to_dict("records")[0]
```

Out[13]:

```
{'film_id': 1,
 'title': 'ACADEMY DINOSAUR',
 'description': 'A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rocki
es',
 'release_year': 2006,
 'language_id': 1,
 'original_language_id': nan,
 'rental_duration': 6,
 'rental_rate': 0.99,
 'length': 86,
 'replacement_cost': 20.99,
 'rating': 'PG',
 'special_features': 'Deleted Scenes,Behind the Scenes',
 'last_update': '2006-02-15 05:03:42',
 'actors': [{'actor_id': 1,
   'film_id': 1,
   'last_update_x': '2006-02-15 05:05:03',
   'first_name': 'PENELOPE',
   'last_name': 'GUINESS',
   'last_update_y': '2006-02-15 04:34:33'},
  {'actor_id': 10,
   'film_id': 1,
   'last_update_x': '2006-02-15 05:05:03',
   'first_name': 'CHRISTIAN',
   'last_name': 'GABLE',
   'last_update_y': '2006-02-15 04:34:33'},
  {'actor_id': 20,
   'film_id': 1,
   'last_update_x': '2006-02-15 05:05:03',
   'first_name': 'LUCILLE',
   'last_name': 'TRACY',
   'last_update_y': '2006-02-15 04:34:33'},
  {'actor_id': 30,
   'film_id': 1,
   'last_update_x': '2006-02-15 05:05:03',
   'first_name': 'SANDRA',
   'last_name': 'PECK',
   'last_update_y': '2006-02-15 04:34:33'},
  {'actor_id': 40,
   'film_id': 1,
   'last_update_x': '2006-02-15 05:05:03',
   'first_name': 'JOHNNY',
   'last_name': 'CAGE',
   'last_update_y': '2006-02-15 04:34:33'},
  {'actor_id': 53,
   'film_id': 1,
   'last_update_x': '2006-02-15 05:05:03',
   'first_name': 'MENA',
   'last_name': 'TEMPLE',
   'last_update_y': '2006-02-15 04:34:33'},
  {'actor_id': 108,
   'film_id': 1,
   'last_update_x': '2006-02-15 05:05:03',
   'first_name': 'WARREN',
   'last_name': 'NOLTE',
   'last_update_y': '2006-02-15 04:34:33'},
  {'actor_id': 162,
   'film_id': 1,
   'last_update_x': '2006-02-15 05:05:03',
   'first_name': 'OPRAH',
   'last_name': 'KILMER',
   'last_update_y': '2006-02-15 04:34:33'},
  {'actor_id': 188,
   'film_id': 1,
   'last_update_x': '2006-02-15 05:05:03',
   'first_name': 'ROCK',
   'last_name': 'DUKAKIS',
   'last_update_y': '2006-02-15 04:34:33'},
  {'actor_id': 198,
   'film_id': 1,
   'last_update_x': '2006-02-15 05:05:03',
   'first_name': 'MARY',
   'last_name': 'KEITEL',
   'last_update_y': '2006-02-15 04:34:33'}],
 'category': [{'film_id': 1, 'category_id': 6, 'name': 'Documentary'}]}
```

In [14]:
```
# Lista de documentos a subir a MongoDB

df_new_films
```

Out[14]:

	film_id	title	description	release_year	language_id	original_language_id	rental_duration	rental_rate	length	repla
0	1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist...	2006	1	NaN	6	0.99	86	
1	2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrat...	2006	1	NaN	3	4.99	48	
2	3	ADAPTATION HOLES	A Astounding Reflection of a Lumberjack And a ...	2006	1	NaN	7	2.99	50	
3	4	AFFAIR PREJUDICE	A Fanciful Documentary of a Frisbee And a Lumb...	2006	1	NaN	5	2.99	117	
4	5	AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef And ...	2006	1	NaN	6	2.99	130	
...	...	...	...	...	...	...	...	...	...	
995	996	YOUNG LANGUAGE	A Unbelieveable Yarn of a Boat And a Database ...	2006	1	NaN	6	0.99	183	
996	997	YOUTH KICK	A Touching Drama of a Teacher And a Cat who mu...	2006	1	NaN	4	0.99	179	
997	998	ZHIVAGO CORE	A Fateful Yarn of a Composer And a Man who mus...	2006	1	NaN	6	0.99	105	
998	999	ZOOLANDER FICTION	A Fateful Reflection of a Waitress And a Boat ...	2006	1	NaN	5	2.99	101	
999	1000	ZORRO ARK	A Intrepid Panorama of a Mad Scientist And a B...	2006	1	NaN	3	4.99	50	

1000 rows × 15 columns

In [15]:
```
# Biblioteca MongoDB
from pymongo import MongoClient
import csv

mongoClient = MongoClient()
```

In [16]:
```
# Creamos una base de datos
#db1 = mongoClient.uv_test1
db = mongoClient.uv_test
```

In [17]:
```
# Insertamos el dataFrame
data_dict = df_new_films.reset_index().to_dict("records")

db.films.insert_many(data_dict)
#db1.films.insert_many(data_dict)
```

Out[17]:
```
<pymongo.results.InsertManyResult at 0x262749813c0>
```

In [18]:
```
pipeline = [{'$unwind': '$actors'},{'$group': {'_id': '$actors.first_name', 'count':{'$sum':1}}}]
pipeline1 = [{'$unwind': '$category'},{'$group': {'_id': '$category.name', 'count':{'$sum':1}}}]
#sd= db.command('aggregate', 'films', pipeline=pipeline1, explain=True)
sd= db.command('aggregate', 'films', pipeline=pipeline1, explain=False)
sd
```

Out[18]:
```
{'cursor': {'firstBatch': [{'_id': 'Classics', 'count': 57},
   {'_id': 'New', 'count': 63},
   {'_id': 'Travel', 'count': 57},
   {'_id': 'Games', 'count': 61},
   {'_id': 'Family', 'count': 69},
   {'_id': 'Children', 'count': 60},
   {'_id': 'Documentary', 'count': 68},
   {'_id': 'Action', 'count': 64},
   {'_id': 'Animation', 'count': 66},
   {'_id': 'Horror', 'count': 56},
   {'_id': 'Music', 'count': 51},
   {'_id': 'Foreign', 'count': 73},
   {'_id': 'Comedy', 'count': 58},
   {'_id': 'Sports', 'count': 74},
   {'_id': 'Sci-Fi', 'count': 61},
   {'_id': 'Drama', 'count': 62}],
  'id': 0,
  'ns': 'uv_test.films'},
 'ok': 1.0}
```

Consultas en MongoDB

Comparación con SQL: https://www.mongodb.com/docs/manual/reference/sql-comparison/

Listar todos los documentos:

```
db.getCollection("films").find({},{})
```

Listar la columna (lista de documentos de actores):

```
db.getCollection("films").find({},{actors:1})
```

```
db.getCollection("films").find({},{"actors.actor_id":1})
```

Funciónes de MapReduce:

```
db.films.aggregate(
  {$unwind: '$actors'},
  {$group: {_id: '$actors.first_name', count:{$sum:1}}}
)
```

Función de agregación:

```
var mapFunction = function() {
  for(var i in this.actors) {
        emit(this.actors[i].first_name, 1);
  }
};


var reduceFunction = function(key, values) {
   return Array.sum(values);
};

 db.getCollection("films").mapReduce(
   mapFunction,
   reduceFunction,
   { out: "map_reduce_example" }
```

```
);

db.map_reduce_example.find().sort( { _id: 1 } )
```

2. Listar ocurrencias del campo categoría

Liste cuantas ocurrencias tienen cada categoria ocupando el framework de MapReduce.

Código

```
var mapFunction = function() {
  for(var i in this.category) {
         emit(this.category[i].name, 1);
  }
};


var reduceFunction = function(key, values) {
   return Array.sum(values);
};

 db.getCollection("films").mapReduce(
   mapFunction,
   reduceFunction,
   { out: "Categorias" }
);

db.Categorias.find().sort( { _id: 1 } )
```

URL al JSON resultante: https://drive.google.com/file/d/1chWUwU0RYway3Suaf_1mNdbzKav0A5fX/view?usp=sharing

3. Crear indice invertido

Para el campo description cree un índice invertido (nueva colección) que entregue los títulos que contengan una palabra en particular.

Código

```
var mapFunction2 = function() {

    for(var i in this.description.split(" ")){
        var key=this.description.split(" ")[i]
        var value=this
        emit(key, value);
    }


};

var reduceFunction2 = function(key, value) {
   return value;
};

db.getCollection("films").mapReduce(
   mapFunction2,
   reduceFunction2,
   { out: "intento1" }
);

db.intento1.find()
```

URL al JSON resultante: https://drive.google.com/file/d/1_273Iby4HUzaXktkOzrZd8CEJNPZqDbC/view?usp=sharing

4. Realizar busqueda en Indice invertido y comentar requerimientos de la consulta

Realice una búsqueda de una palabra (existe en la descripción) en el índice invertido y verifique el número de documentos que examino con .explain("executionStats").

Código

```
db.getCollection("intento1").find({_id: "Berlin"}).explain("executionStats")
```

Respuesta:


```
{
    "explainVersion" : "1",
    "queryPlanner" : {
        "namespace" : "uv_test.intento1",
        "indexFilterSet" : false,
        "parsedQuery" : {
            "_id" : {
                "$eq" : "Berlin"
            }
        },
        "queryHash" : "740C02B0",
        "planCacheKey" : "E351FFEC",
        "maxIndexedOrSolutionsReached" : false,
        "maxIndexedAndSolutionsReached" : false,
        "maxScansToExplodeReached" : false,
        "winningPlan" : {
            "stage" : "IDHACK"
        },
        "rejectedPlans" : [

        ]
    },
    "executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 1.0,
        "executionTimeMillis" : 0.0,
        "totalKeysExamined" : 1.0,
        "totalDocsExamined" : 1.0,
        "executionStages" : {
            "stage" : "IDHACK",
            "nReturned" : 1.0,
            "executionTimeMillisEstimate" : 0.0,
            "works" : 2.0,
            "advanced" : 1.0,
            "needTime" : 0.0,
            "needYield" : 0.0,
            "saveState" : 0.0,
            "restoreState" : 0.0,
            "isEOF" : 1.0,
            "keysExamined" : 1.0,
            "docsExamined" : 1.0
        }
    },
    "command" : {
        "find" : "intento1",
        "filter" : {
            "_id" : "Berlin"
        },
        "$db" : "uv_test"
    },
    "serverInfo" : {
        "host" : "DESKTOP-3L6OHEQ",
        "port" : 27017.0,
```

```
        "version" : "6.0.0",
        "gitVersion" : "e61bf27c2f6a83fed36e5a13c008a32d563babe2"
    },
    "serverParameters" : {
        "internalQueryFacetBufferSizeBytes" : 104857600.0,
        "internalQueryFacetMaxOutputDocSizeBytes" : 104857600.0,
        "internalLookupStageIntermediateDocumentMaxSizeBytes" : 104857600.0,
        "internalDocumentSourceGroupMaxMemoryBytes" : 104857600.0,
        "internalQueryMaxBlockingSortMemoryUsageBytes" : 104857600.0,
        "internalQueryProhibitBlockingMergeOnMongoS" : 0.0,
        "internalQueryMaxAddToSetBytes" : 104857600.0,
        "internalDocumentSourceSetWindowFieldsMaxMemoryBytes" : 104857600.0
    },
    "ok" : 1.0
}
```


Comentario: Al buscar la palarba "Berlín" dentro del indice invertido, la consulta se ejecutó correctamente, se analizó solo 1 documento, se retornó 1 elemento con un tiempo de busqueda de 0 milisegundos.

5. Realizar busqueda en el campo descripcion y comentar requerimientos de la consulta

Busque una palabra en el campo descripción y liste el número de documentos que examino. **Código**

```
db.getCollection("films").find({description:/Berlin/},{}).explain("executionStats")
```

Respuesta:


```
{
    "explainVersion" : "1",
    "queryPlanner" : {
        "namespace" : "uv_test.films",
        "indexFilterSet" : false,
        "parsedQuery" : {
            "description" : {
                "$regex" : "Berlin"
            }
        },
        "queryHash" : "E61A6F7B",
        "planCacheKey" : "E61A6F7B",
        "maxIndexedOrSolutionsReached" : false,
        "maxIndexedAndSolutionsReached" : false,
        "maxScansToExplodeReached" : false,
        "winningPlan" : {
            "stage" : "COLLSCAN",
            "filter" : {
                "description" : {
                    "$regex" : "Berlin"
                }
            },
            "direction" : "forward"
        },
        "rejectedPlans" : [

        ]
    },
    "executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 53.0,
        "executionTimeMillis" : 1.0,
        "totalKeysExamined" : 0.0,
        "totalDocsExamined" : 1000.0,
        "executionStages" : {
```

```
                "stage" : "COLLSCAN",
                "filter" : {
                    "description" : {
                        "$regex" : "Berlin"
                    }
                },
                "nReturned" : 53.0,
                "executionTimeMillisEstimate" : 0.0,
                "works" : 1002.0,
                "advanced" : 53.0,
                "needTime" : 948.0,
                "needYield" : 0.0,
                "saveState" : 1.0,
                "restoreState" : 1.0,
                "isEOF" : 1.0,
                "direction" : "forward",
                "docsExamined" : 1000.0
            }
        },
        "command" : {
            "find" : "films",
            "filter" : {
                "description" : /Berlin/
            },
            "projection" : {

            },
            "$db" : "uv_test"
        },
        "serverInfo" : {
            "host" : "DESKTOP-3L6OHEQ",
            "port" : 27017.0,
            "version" : "6.0.0",
            "gitVersion" : "e61bf27c2f6a83fed36e5a13c008a32d563babe2"
        },
        "serverParameters" : {
            "internalQueryFacetBufferSizeBytes" : 104857600.0,
            "internalQueryFacetMaxOutputDocSizeBytes" : 104857600.0,
            "internalLookupStageIntermediateDocumentMaxSizeBytes" : 104857600.0,
            "internalDocumentSourceGroupMaxMemoryBytes" : 104857600.0,
            "internalQueryMaxBlockingSortMemoryUsageBytes" : 104857600.0,
            "internalQueryProhibitBlockingMergeOnMongoS" : 0.0,
            "internalQueryMaxAddToSetBytes" : 104857600.0,
            "internalDocumentSourceSetWindowFieldsMaxMemoryBytes" : 104857600.0
        },
        "ok" : 1.0
    }
```


Comentario: Al buscar la palarba "Berlín" directamente dentro del campo descripción y listar el numero de documentos, la consulta se ejecutó correctamente, se analizaon 1000 docuemntos, retornando 53 elemento con un tiempo de busqueda de 1 milisegundos.

6. Crear indice de texto, realizar busqueda en el campo descripcion y comentar requerimientos de la consulta

Cree un índice de texto para la llave description con la sentencia .createIndex(), realice una búsqueda y liste nuevamente los documentos que examino.

Código

```
db.getCollection("films").createIndex({description:1})
```

```
db.getCollection("films").find({description:/Berlin/},{}).explain("executionStats")
```

Resultado:


```
{
    "numIndexesBefore" : 2.0,
    "numIndexesAfter" : 2.0,
    "note" : "all indexes already exist",
    "ok" : 1.0
}
{
    "explainVersion" : "1",
    "queryPlanner" : {
        "namespace" : "uv_test.films",
        "indexFilterSet" : false,
        "parsedQuery" : {
            "description" : {
                "$regex" : "Berlin"
            }
        },
        "queryHash" : "E61A6F7B",
        "planCacheKey" : "EA7DECD1",
        "maxIndexedOrSolutionsReached" : false,
        "maxIndexedAndSolutionsReached" : false,
        "maxScansToExplodeReached" : false,
        "winningPlan" : {
            "stage" : "FETCH",
            "inputStage" : {
                "stage" : "IXSCAN",
                "filter" : {
                    "description" : {
                        "$regex" : "Berlin"
                    }
                },
                "keyPattern" : {
                    "description" : 1.0
                },
                "indexName" : "description_1",
                "isMultiKey" : false,
                "multiKeyPaths" : {
                    "description" : [

                    ]
                },
                "isUnique" : false,
                "isSparse" : false,
                "isPartial" : false,
                "indexVersion" : 2.0,
                "direction" : "forward",
                "indexBounds" : {
                    "description" : [
                        "[\"\", {})",
                        "[/Berlin/, /Berlin/]"
                    ]
                }
            }
        },
        "rejectedPlans" : [

        ]
    },
    "executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 53.0,
        "executionTimeMillis" : 18.0,
        "totalKeysExamined" : 1000.0,
        "totalDocsExamined" : 53.0,
        "executionStages" : {
```

```
            "stage" : "FETCH",
            "nReturned" : 53.0,
            "executionTimeMillisEstimate" : 0.0,
            "works" : 1001.0,
            "advanced" : 53.0,
            "needTime" : 947.0,
            "needYield" : 0.0,
            "saveState" : 1.0,
            "restoreState" : 1.0,
            "isEOF" : 1.0,
            "docsExamined" : 53.0,
            "alreadyHasObj" : 0.0,
            "inputStage" : {
                "stage" : "IXSCAN",
                "filter" : {
                    "description" : {
                        "$regex" : "Berlin"
                    }
                },
                "nReturned" : 53.0,
                "executionTimeMillisEstimate" : 0.0,
                "works" : 1001.0,
                "advanced" : 53.0,
                "needTime" : 947.0,
                "needYield" : 0.0,
                "saveState" : 1.0,
                "restoreState" : 1.0,
                "isEOF" : 1.0,
                "keyPattern" : {
                    "description" : 1.0
                },
                "indexName" : "description_1",
                "isMultiKey" : false,
                "multiKeyPaths" : {
                    "description" : [

                    ]
                },
                "isUnique" : false,
                "isSparse" : false,
                "isPartial" : false,
                "indexVersion" : 2.0,
                "direction" : "forward",
                "indexBounds" : {
                    "description" : [
                        "[\"\", {})",
                        "[/Berlin/, /Berlin/]"
                    ]
                },
                "keysExamined" : 1000.0,
                "seeks" : 1.0,
                "dupsTested" : 0.0,
                "dupsDropped" : 0.0
            }
        }
    },
    "command" : {
        "find" : "films",
        "filter" : {
            "description" : /Berlin/
        },
        "projection" : {

        },
        "$db" : "uv_test"
    },
```

```
        "serverInfo" : {
            "host" : "DESKTOP-3L6OHEQ",
            "port" : 27017.0,
            "version" : "6.0.0",
            "gitVersion" : "e61bf27c2f6a83fed36e5a13c008a32d563babe2"
        },
        "serverParameters" : {
            "internalQueryFacetBufferSizeBytes" : 104857600.0,
            "internalQueryFacetMaxOutputDocSizeBytes" : 104857600.0,
            "internalLookupStageIntermediateDocumentMaxSizeBytes" : 104857600.0,
            "internalDocumentSourceGroupMaxMemoryBytes" : 104857600.0,
            "internalQueryMaxBlockingSortMemoryUsageBytes" : 104857600.0,
            "internalQueryProhibitBlockingMergeOnMongoS" : 0.0,
            "internalQueryMaxAddToSetBytes" : 104857600.0,
            "internalDocumentSourceSetWindowFieldsMaxMemoryBytes" : 104857600.0
        },
        "ok" : 1.0
    }
```


Comentario: Al buscar la palarba "Berlín" dentro del indice de texto creado con el campo descripción, la consulta se ejecutó correctamente, se analizaon 53 docuemntos, retornando 53 elemento con un tiempo de busqueda de 18 milisegundos

7. Conclusiones

Concluya brevemente el porque de las diferencias en el número de documentos examinados entre el punto 4, 5 y 6.

Se concluye que la forma más eficiente de realizar una busqueda, considerando las consultas realizas, es a través del indice invertido, ya que solo requirió analizar 1 documento, a diferencia de buscar directamente en el campo, la consulta tuvo que analizar los 1000 registros existentes dentro del campo. Por otro lado, La busqueda a través de un índice de texto, revisa solo 53 docuemtnos pero con un tiempo de ejecución más de 50 veces mayor a las otras busquedas.

In [ ]: