A Python Code

A .1 Creation of one single dataset from the tsv imdb file

```python

import pandas as pd
from google.colab import drive
drive.mount('/content/drive')
numberofrows=None #100000
title_basics = pd.read_csv("/content/drive/MyDrive/Dataset/Original/title_basics.tsv",sep='\t',nrows=numberofrows,header=0)
title_principals = pd.read_csv("/content/drive/MyDrive/Dataset/Original/title_principals.tsv",nrows=numberofrows,sep='\t',header=0)

keep_col = ["tconst","titleType","primaryTitle","originalTitle","startYear","runtimeMinutes","genres"]
title_basics = title_basics[keep_col]
title_basics = title_basics[title_basics["titleType"].str.contains("movie") == True]

print(title_basics.head(3))

merged1=pd.merge(title_basics,title_principals,how='inner',on='tconst')
del title_basics,title_principals
print(merged1)

name_basics=pd.read_csv("/content/drive/MyDrive/Dataset/Original/name_basics.tsv",sep='\t',nrows=numberofrows,header=0)

merged2=pd.merge(merged1,name_basics,how='inner',on='nconst')
del merged1
merged2=merged2.drop(columns=["ordering","nconst","birthYear","deathYear","knownForTitles","primaryProfession"])
del name_basics
print(merged2)

category=merged2.groupby('tconst')['category'].apply(list).reset_index(name='category')
job=merged2.groupby('tconst')['job'].apply(list).reset_index(name='job')
characters=merged2.groupby('tconst')['characters'].apply(list).reset_index(name='characters')
primaryName=merged2.groupby('tconst')['primaryName'].apply(list).reset_index(name='primaryName')
result=merged2.drop_duplicates(subset=['tconst'])
result=result.drop(['category'], axis=1).drop(['job'], axis=1).drop(['characters'], axis=1).drop(['primaryName'], axis=1)
result=result.merge(category,on='tconst').merge(job,on='tconst').merge(characters,on='tconst').merge(primaryName,on='tconst')
print(result)

result.to_csv("/content/drive/MyDrive/Dataset/resultSetFinale.csv",index=False)
```

Listing 1: Test

A .2 Creation of one single dataset from the csv kaggle file

```

```

```python
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')
numberofrows=None #100000
movies = pd.read_csv("/content/drive/MyDrive/Dataset/Original/rotten_movies.csv",nrows=numberofrows,header=0)
reviews = pd.read_csv("/content/drive/MyDrive/Dataset/Original/rotten_reviews.csv",nrows=numberofrows,header=0)
to_keep = ["rotten_tomatoes_link", "movie_title", "production_company","critics_consensus",
           "tomatometer_status", "tomatometer_rating", "tomatometer_count",
           "audience_status", "audience_rating", "audience_count",
           "tomatometer_top_critics_count", "tomatometer_fresh_critics_count",
           "tomatometer_rotten_critics_count"]

movies = movies[to_keep]
to_drop = ["publisher_name"]
reviews=reviews.drop(columns=to_drop)

merged=pd.merge(movies,reviews,how='inner',on="rotten_tomatoes_link")
print(merged)

categories = {}
arr = ["critic_name", "top_critic", "review_type", "review_score", "review_date", "review_content"]
for x in arr:
  categories[x]=merged.groupby('rotten_tomatoes_link')[x].apply(list).reset_index(name=x)

result=merged.drop_duplicates(subset=['rotten_tomatoes_link'])
for x in arr:
  result=result.drop([x], axis=1)
for x in arr:
  result=result.merge(categories[x],on='rotten_tomatoes_link')

print(result)

result.to_csv("/content/drive/MyDrive/Dataset/resultSetRotten.csv",index=False)
```

Listing 2: Test

A .3 Merging of the file generated in the previous script

```python

import pandas as pd
from google.colab import drive
drive.mount('/content/drive')
numberofrows=None
imdb = pd.read_csv("/content/drive/MyDrive/Dataset/resultSetFinale.csv",nrows=numberofrows,header=0)
rotten = pd.read_csv("/content/drive/MyDrive/Dataset/resultSetRotten.csv",nrows=numberofrows,header=0)

merged = {}
choose = ['primaryTitle', 'originalTitle']
rowHeadDataset = 20
for x in choose:
  merged[x]=pd.merge(imdb,rotten,how='inner', left_on=x, right_on='movie_title')
  print(x)
  merged[x]=merged[x].drop_duplicates(subset=[x])
  merged[x]=merged[x].drop(columns=['titleType','tomatometer_count','tomatometer_top_critics_count'])
  merged[x]=merged[x].rename(columns={'startYear':'year'})
  merged[x]=merged[x].drop(columns=['rotten_tomatoes_link', 'movie_title']+[j for j in choose if j!=x])
  print(len(pd.unique(merged[x][x])))
  print(list(merged[x]))
  print("==========================")
  merged[x].to_csv(f"/content/drive/MyDrive/Dataset/ImdbJoinRotten{x}.csv", index=False)
  merged[x]=merged[x].head(rowHeadDataset)
  merged[x].to_csv(f"/content/drive/MyDrive/Dataset/headDataset{x}.csv",index=False)
```

Listing 3: Test

A .4 Collapsing different rows in a single one generating an array for personnel field

```python

import pandas as pd
from ast import literal_eval
from google.colab import drive
drive.mount('/content/drive')

numberofrows=None
df = pd.read_csv("/content/drive/MyDrive/Dataset/ImdbJoinRottenprimaryTitle.csv",nrows=numberofrows,header=0)

#print([x.split(',') for x in df['genres']])
print(df)

col = ["primaryName","category","job","characters"]
col1 = ["critic_name","top_critic","review_type","review_score","review_date", "review_content"]

df['personnel'] = ""
df['review'] = ""

for row in range(df[col[0]].size):
  it = df['genres'][row]
  df['genres'][row] = ['"' + x + '"' for x in it.split(',')] if it != '\\N' else []
  tmp = []
  for c in col:
    tmp.append({c:eval(df[c][row])})
  res = []
  for c in range(len(tmp[0][col[0]])):
    res.append({})
  for i, j in zip(col, tmp):
    for idx, x in enumerate(j[i]):
      #print(i, idx, x)
      if x != '\\N':
        if i == 'characters':
          x = eval(x)
          res[idx]["'" + i + "'"] = "'" + str(x).replace("'", "##single-quote##").replace('"', "##double-quote##") + "'"
    df['personnel'][row] = list(res)
    #print(res)
    ###
    tmp = []
    for c in col1:
      to_eval = df[c][row].replace('nan', 'None')
      arr = eval(to_eval)
      if c == "review_date":
        for i, elem in enumerate(arr):
          arr[i] = elem + "T00:00:00.000+00:00"
      tmp.append({c:arr})
      #print(tmp)
    res = []
    for c in range(len(tmp[0][col1[0]])):
      res.append({})
    for i, j in zip(col1, tmp):
      for idx, x in enumerate(j[i]):
        #print(i, idx, x)
        if x != '\\N':
          res[idx]["'" + i + "'"] = "'" + str(x).replace("True", "true").replace("False", "false").replace("'", "##single-quote##").replace('"', "##double-quote##") + "'"
    df['review'][row] = list(res)
    #df['review'][row] = eval(str(res))
    #print(res)
    #print()

df=df.drop(columns=col)
df=df.drop(columns=col1)
df=df.drop(columns=['tconst'])

print(df["review"][0])

it = df['personnel'][0]#[4]['review_content']
print(type(it))
print(it)

df.to_csv("/content/drive/MyDrive/Dataset/movieCollectionEmbeddedReviewPersonnel.csv",index=False)
df = df.head(20)
df.to_csv("/content/drive/MyDrive/Dataset/headmovieCollectionEmbeddedReviewPersonnel.csv",index=False)
```

Listing 4: Test

A .5 Generates a hashed password for all the users

```
import hashlib
#from pprint import pprint as print
from pymongo import MongoClient

def get_database():
    CONNECTION_STRING = "mongodb://localhost:27017"
    client = MongoClient(CONNECTION_STRING)
    return client['rottenMovies']

if __name__ == "__main__":
    dbname = get_database()
    collection = dbname['user']
    total = collection.count_documents({})
    for i, user in enumerate(collection.find()):
        all_reviews = user['last_3_reviews']
        sorted_list = sorted(all_reviews, key=lambda t: t['review_date'])[-3:]

        hashed = hashlib.md5(user["username"].encode()).hexdigest()

        newvalues = { "$set": { 'password': hashed, 'last_3_reviews': sorted_list } }
        filter = { 'username': user['username']}
        collection.update_one(filter, newvalues)
        print(f"{i/total:%}\r", end='')
    print()
```

Listing 5: Test

A .6 Generates the graph database

```
from pymongo import MongoClient
from neo4j import GraphDatabase
from random import randint, shuffle

def get_database():
    CONNECTION_STRING = "mongodb://localhost:27017"
    client = MongoClient(CONNECTION_STRING)
    return client['rottenMovies']

class Neo4jGraph:

    def __init__(self, uri, user, password):
        self.driver = GraphDatabase.driver(uri, auth=(user, password), database="rottenmoviesgraphdb")

    def close(self):
        self.driver.close()

    def addUser(self, uid, name, isTop):
        with self.driver.session() as session:
            if isTop:
                result = session.execute_write(self._addTopCritic, uid, name)
            else:
                result = session.execute_write(self._addUser, uid, name)

    def addMovie(self, mid, title):
        with self.driver.session() as session:
            result = session.execute_write(self._addMovie, mid, title)

    def addReview(self, name, mid, freshness, content, date):
        with self.driver.session() as session:
            result = session.execute_write(self._addReview, name, mid, freshness, content, date)

    def addFollow(self, uid, cid):
        with self.driver.session() as session:
            result = session.execute_write(self._addFollow, uid, cid)

```

```python
    @staticmethod
    def _addUser(tx, uid, name):
        query = "CREATE (n:User{id:\"" + str(uid) + "\", name:\"" + name.replace('"', '\\"') + "\"})"
        #print(query)
        result = tx.run(query)

    @staticmethod
    def _addTopCritic(tx, cid, name):
        query = "CREATE(m:TopCritic{id:\"" + str(cid) + "\", name:\"" + name.replace('"', '\\"') + "\"})"
        #print(query)
        result = tx.run(query)

    @staticmethod
    def _addMovie(tx, mid, title):
        query = "CREATE(o:Movie{id:\"" + str(mid) + "\", title:\"" + title.replace('"', '\\"') + "\"})"
        #print(query)
        result = tx.run(query)

    @staticmethod
    def _addReview(tx, name, mid, freshness, content, date): # date in format YYYY-mm-dd, freshness in [TRUE, FALSE]
        query = "MATCH(n{name:\"" + str(name).replace('"', '\\"') + "\"}), (m:Movie{id:\"" + str(mid) + "\"}) CREATE (n)-[r:REVIEWED{freshness:" + freshness + ", date:date('" + date + "'), content:\"" + content.replace('"', '\\"') + "\"}]->(m)"
        #print(query)
        result = tx.run(query)

    @staticmethod
    def _addFollow(tx, uid, cid):
        query = "MATCH(n:User{id:\"" + str(uid) + "\"}), (m:TopCritic{id:\"" + str(cid) + "\"}) CREATE (n)-[r:FOLLOWS]->(m)"
        #print(query)
        result = tx.run(query)

if __name__ == "__main__":
    # dbs initialization
    dbname = get_database()
    graphDB = Neo4jGraph("bolt://localhost:7687", "neo4j", "password")

    # user creation
    collection = dbname['user']
    total = collection.count_documents({})
    print(f"user {total = }")
    for i, user in enumerate(list(collection.find({}, {"_id":1, "username":1, "date_of_birth":1}))):
        graphDB.addUser(user['_id'], user['username'], 'date_of_birth' not in user)
        if not i%100:
            print(f"{(i+1)/total:%}\r", end='')

    # movie creation and review linking
    collection = dbname['movie']
    total = collection.count_documents({})
    print(f"\nmovie {total = }")
    for i, movie in enumerate(list(collection.find({}, {"_id":1, "primaryTitle":1, "review":1}))):
        graphDB.addMovie(movie['_id'], movie['primaryTitle'])
        movie['review'] = list({v['critic_name']:v for v in movie['review']}.values()) # make unique reviews per critic
        for rev in movie['review']:
            graphDB.addReview(rev['critic_name'], movie['_id'], {"Fresh":"TRUE", "Rotten":"FALSE"}[rev['review_type']], str(rev['review_content'])[:15], str(rev['review_date'])[:10])
        print(f"{(i+1)/total:%}\r", end='')

    # follow linking
    collection = dbname['user']
    uids = [x['_id'] for x in list(collection.find({"date_of_birth":{"$exists":True}}, {"_id":1}))]
    cids = [x['_id'] for x in list(collection.find({"date_of_birth":{"$exists":False}}, {"_id":1}))]
    total = len(uids)
    print(f"\nfollow {total = }")
    for i, user in enumerate(uids):
        shuffle(cids)
        for j in range(randint(0, 20)):
            graphDB.addFollow(user, cids[j])
        print(f"{i/total:%}\r", end='')

    graphDB.close()
```
Listing 6: Test

B Mongosh scripts

B .1 Perform the escape on the string fields

```

db.movie.find().forEach(
    x => {
        print(x.primaryTitle);
        x.review = JSON.parse(
            x.review.replaceAll('"\'', '"')
                .replaceAll('\'"', '"')
                .replaceAll('"false"', 'false')
                .replaceAll('"true"', 'true')
                .replaceAll('"None"', 'null')
                .replaceAll(/\\x\d{2}/g, "")
                .replaceAll("##single-quote##", "\'")
                .replaceAll("##double-quote##", '\\"')
                .replaceAll("\\x", "x")
        );
        x.personnel = JSON.parse(
            x.personnel.replaceAll('"\'', '"')
                .replaceAll('\'"', '"')
                .replaceAll('"None"', 'null')
                .replaceAll("##single-quote##", '\'')
                .replaceAll("##double-quote##", '\\"')
                .replaceAll('"[\'', '["')
                .replaceAll('"[\\"', '["')
                .replaceAll('\']"', '"]')
                .replaceAll('\\"]"', '"]')
                .replaceAll(/(\[[^[:]*)\\", \\"([^]:]*\])/g, '$1", "$2')
                .replaceAll(/(\[[^[:]*)\', \\"([^]:]*\])/g, '$1", "$2')
                .replaceAll(/(\[[^[:]*)\\", \'([^]:]*\])/g, '$1", "$2')
        );
        x.genres = JSON.parse(
            x.genres = x.genres.replaceAll('"\'', '"')
                    .replaceAll('\'"', '"')
                    .replaceAll('"None"', 'null')
                    .replaceAll("##single-quote##", "\'")
                    .replaceAll("##double-quote##", '\\"')
        );
        db.movie.updateOne(
            {"_id": x._id},
            {$set:
                {
                    "review": x.review,
                    "personnel": x.personnel,
                    "genres": x.genres,
                    "runtimeMinutes":parseInt(x.runtimeMinutes),
                    "year":parseInt(x.year),
                    "tomatometer_rating":parseFloat(x.tomatometer_rating),
                    "audience_rating":parseFloat(x.audience_rating),
                    "audience_count":parseFloat(x.audience_count),
                    "tomatometer_fresh_critics_count":parseInt(x.tomatometer_fresh_critics_count),
                    "tomatometer_rotten_critics_count":parseInt(x.tomatometer_rotten_critics_count)
                }
            }
        );
    }
);
```

Listing 7: Test

B .2 Normalize the date field in the DB

```
total = db.movie.find().count();
i = 0;
db.movie.find().forEach(
    x => {
        print(x.primaryTitle);
        x.review.forEach(rev =>{
            if(typeof (rev.review_date) === "string" ){
                db.movie.updateOne(
                    {primaryTitle: x.primaryTitle },
                    { $set: { "review.$[elem].review_date" : new Date(rev.
   review_date) } },
                    { arrayFilters: [ { "elem.critic_name": rev.critic_name }
    ] }
                )
            }
        })
        print(100*i++/total);
});
```

Listing 8: Test

B .3 Create a new collection for the user based on the data present in the movie collection

```
total = db.runCommand({ distinct: "movie", key: "review.critic_name", query:
   {"review.critic_name":{$ne:null}}}).values.length
i = 0;
db.runCommand(
{ distinct: "movie", key: "review.critic_name", query: {"review.critic_name"
   :{$ne:null}}}).values.forEach(
    (x) => {
        review_arr = []
        movie_arr = []
        is_top = false
        db.movie.aggregate(
            [
                { $project:
                    {
                        index: { $indexOfArray: ["$review.critic_name", x]},
                        primaryTitle: 1
                    }},
                {$match:{index:{$gt:-1}}}
            ]
        ).forEach(
            y => {
                tmp = db.movie.aggregate([
                    {
                        $project:
                        {
                            top_critic: {
```

```
                            $arrayElemAt: ["$review.top_critic", y.index]
                        },
                        primaryTitle: y.primaryTitle,
                        review_type: {
                            $arrayElemAt: ["$review.review_type", y.index]
                        },
                        review_score: {
                            $arrayElemAt: ["$review.review_score", y.index]
                        },
                        review_date: {
                            $arrayElemAt: ["$review.review_date", y.index]
                        },
                        review_content: {
                            $arrayElemAt: ["$review.review_content", y.index]
                        }
                    }
                },
                {
                    $match:{_id:{$eq:y._id}}
                }
            ]).toArray()[0];
            is_top |= tmp.top_critic;
            review_arr.push(tmp)
            //movie_arr.push(tmp._id)
            movie_arr.push({"movie_id": tmp._id, "primaryTitle": y.primaryTitle, "review_index": y.index})
        })

    name_parts = x.split(/\s/)
    first_name = name_parts.splice(0, 1)[0]
    last_name = name_parts.join(' ')

    print(100*i++/total, x, is_top)
    //print(first_name, ':', last_name)
    //print(review_arr)
    //print(movie_arr)
    db.user.insertOne(
        {
            "username": x,
            "password": "",
            "first_name": first_name,
            "last_name": last_name,
            "registration_date": new Date("2000-01-01"),
            "last_3_reviews": review_arr,
            "reviews" : movie_arr
        }
    );
    if (!is_top){
        db.user.updateOne(
            {"username": x},
            {$set:
                {"date_of_birth": new Date("1970-07-20")}
            }
        )
    }
    print("================================")
}
```

```
)
```

Listing 9: Test

C MongoDB indexes:Movie collection

C .1 primaryTitle

Before the index

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'rottenMovies.movie',
    indexFilterSet: false,
    parsedQuery: { primaryTitle: { '$eq': 'Evidence' } },
    queryHash: '9839850C',
    planCacheKey: '9839850C',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'COLLSCAN',
      filter: { primaryTitle: { '$eq': 'Evidence' } },
      direction: 'forward'
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 1,
    executionTimeMillis: 275,
    totalKeysExamined: 0,
    totalDocsExamined: 14104,
    executionStages: {
      stage: 'COLLSCAN',
      filter: { primaryTitle: { '$eq': 'Evidence' } },
      nReturned: 1,
      executionTimeMillisEstimate: 245,
      works: 14106,
      advanced: 1,
      needTime: 14104,
      needYield: 0,
      saveState: 18,
      restoreState: 18,
      isEOF: 1,
      direction: 'forward',
      docsExamined: 14104
    }
  },
  command: {
    find: 'movie',
    filter: { primaryTitle: 'Evidence' },
    '$db': 'rottenMovies'
  },
  serverInfo: {
    host: 'Profile2022LARGE10',
    port: 27017,
    version: '6.0.3',
    gitVersion: 'f803681c3ae19817d31958965850193de067c516'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1673280853, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "
    hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1673280853, i: 1 })
}
```

Listing 10: Test

After the index

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'rottenMovies.movie',
    indexFilterSet: false,
    parsedQuery: { primaryTitle: { '$eq': 'Evidence' } },
    queryHash: '9839850C',
    planCacheKey: 'B734708E',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { primaryTitle: 1 },
        indexName: 'primaryTitle_1',
        isMultiKey: false,
        multiKeyPaths: { primaryTitle: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { primaryTitle: [ '["Evidence", "Evidence"]' ] }
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 1,
    executionTimeMillis: 1,
    totalKeysExamined: 1,
    totalDocsExamined: 1,
    executionStages: {
      stage: 'FETCH',
      nReturned: 1,
      executionTimeMillisEstimate: 0,
      works: 2,
      advanced: 1,
      needTime: 0,
      needYield: 0,
      saveState: 0,
      restoreState: 0,
      isEOF: 1,
      docsExamined: 1,
      alreadyHasObj: 0,
      inputStage: {
        stage: 'IXSCAN',
        nReturned: 1,
        executionTimeMillisEstimate: 0,
        works: 2,
        advanced: 1,
        needTime: 0,
        needYield: 0,
        saveState: 0,
        restoreState: 0,
        isEOF: 1,
        keyPattern: { primaryTitle: 1 },
        indexName: 'primaryTitle_1',
        isMultiKey: false,
        multiKeyPaths: { primaryTitle: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { primaryTitle: [ '["Evidence", "Evidence"]' ] },
        keysExamined: 1,
        seeks: 1,
        dupsTested: 0,
        dupsDropped: 0
      }
    }
  },
  command: {
    find: 'movie',
    filter: { primaryTitle: 'Evidence' },
    '$db': 'rottenMovies'
  },
  serverInfo: {
    host: 'Profile2022LARGE10',
    port: 27017,
    version: '6.0.3',
    gitVersion: 'f803681c3ae19817d31958965850193de067c516'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1673285103, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "
   hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1673285103, i: 1 })
}
```

Listing 11: Test

C .2 year

Before the index

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'rottenMovies.movie',
    indexFilterSet: false,
    parsedQuery: { year: { '$eq': 2012 } },
    queryHash: '412E8B51',
    planCacheKey: '412E8B51',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'COLLSCAN',
      filter: { year: { '$eq': 2012 } },
      direction: 'forward'
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 480,
    executionTimeMillis: 13,
    totalKeysExamined: 0,
    totalDocsExamined: 14104,
    executionStages: {
      stage: 'COLLSCAN',
      filter: { year: { '$eq': 2012 } },
      nReturned: 480,
      executionTimeMillisEstimate: 1,
      works: 14106,
      advanced: 480,
      needTime: 13625,
      needYield: 0,
      saveState: 14,
      restoreState: 14,
      isEOF: 1,
      direction: 'forward',
      docsExamined: 14104
    }
  },
  command: { find: 'movie', filter: { year: 2012 }, '$db': 'rottenMovies' },
  serverInfo: {
    host: 'Profile2022LARGE10',
    port: 27017,
    version: '6.0.3',
    gitVersion: 'f803681c3ae19817d31958965850193de067c516'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1673280923, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1673280923, i: 1 })
}
```

Listing 12: Test

After the index

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'rottenMovies.movie',
    indexFilterSet: false,
    parsedQuery: { year: { '$eq': 2012 } },
    queryHash: '412E8B51',
    planCacheKey: '62915BA3',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { year: 1 },
        indexName: 'year_1',
        isMultiKey: false,
        multiKeyPaths: { year: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { year: [ '[2012, 2012]' ] }
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 480,
    executionTimeMillis: 2,
    totalKeysExamined: 480,
    totalDocsExamined: 480,
    executionStages: {
      stage: 'FETCH',
      nReturned: 480,
      executionTimeMillisEstimate: 0,
      works: 481,
      advanced: 480,
      needTime: 0,
      needYield: 0,
      saveState: 0,
      restoreState: 0,
      isEOF: 1,
      docsExamined: 480,
      alreadyHasObj: 0,
      inputStage: {
        stage: 'IXSCAN',
        nReturned: 480,
        executionTimeMillisEstimate: 0,
        works: 481,
        advanced: 480,
        needTime: 0,
        needYield: 0,
        saveState: 0,
        restoreState: 0,
        isEOF: 1,
        keyPattern: { year: 1 },
        indexName: 'year_1',
        isMultiKey: false,
        multiKeyPaths: { year: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { year: [ '[2012, 2012]' ] },
        keysExamined: 480,
        seeks: 1,
        dupsTested: 0,
        dupsDropped: 0
      }
    }
  },
  command: { find: 'movie', filter: { year: 2012 }, '$db': 'rottenMovies' },
  serverInfo: {
    host: 'Profile2022LARGE10',
    port: 27017,
    version: '6.0.3',
    gitVersion: 'f803681c3ae19817d31958965850193de067c516'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1673285143, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "
    hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1673285143, i: 1 })
}
```

Listing 13: Test

C .3 top critic rating

Before the index

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'rottenMovies.movie',
    indexFilterSet: false,
    parsedQuery: {},
    queryHash: '33018E32',
    planCacheKey: '33018E32',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'SORT',
      sortPattern: { top_critic_rating: 1 },
      memLimit: 104857600,
      type: 'simple',
      inputStage: { stage: 'COLLSCAN', direction: 'forward' }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 14104,
    executionTimeMillis: 1818,
    totalKeysExamined: 0,
    totalDocsExamined: 14104,
    executionStages: {
      stage: 'SORT',
      nReturned: 14104,
      executionTimeMillisEstimate: 1740,
      works: 28211,
      advanced: 14104,
      needTime: 14106,
      needYield: 0,
      saveState: 47,
      restoreState: 47,
      isEOF: 1,
      sortPattern: { top_critic_rating: 1 },
      memLimit: 104857600,
      type: 'simple',
      totalDataSizeSorted: 262640385,
      usedDisk: true,
      spills: 3,
      inputStage: {
        stage: 'COLLSCAN',
        nReturned: 14104,
        executionTimeMillisEstimate: 0,
        works: 14106,
        advanced: 14104,
        needTime: 1,
        needYield: 0,
        saveState: 47,
        restoreState: 47,
        isEOF: 1,
        direction: 'forward',
        docsExamined: 14104
      }
    }
  },
  command: {
    find: 'movie',
    filter: {},
    sort: { top_critic_rating: 1 },
    '$db': 'rottenMovies'
  },
  serverInfo: {
    host: 'Profile2022LARGE10',
    port: 27017,
    version: '6.0.3',
    gitVersion: 'f803681c3ae19817d31958965850193de067c516'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1673287293, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "
    hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1673287293, i: 1 })
}
```

Listing 14: Test

After the index

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'rottenMovies.movie',
    indexFilterSet: false,
    parsedQuery: {},
    queryHash: '33018E32',
    planCacheKey: '33018E32',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { top_critic_rating: 1 },
        indexName: 'top_critic_rating_1',
        isMultiKey: false,
        multiKeyPaths: { top_critic_rating: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { top_critic_rating: [ '[MinKey, MaxKey]' ] }
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 14104,
    executionTimeMillis: 24,
    totalKeysExamined: 14104,
    totalDocsExamined: 14104,
    executionStages: {
      stage: 'FETCH',
      nReturned: 14104,
      executionTimeMillisEstimate: 5,
      works: 14105,
      advanced: 14104,
      needTime: 0,
      needYield: 0,
      saveState: 14,
      restoreState: 14,
      isEOF: 1,
      docsExamined: 14104,
      alreadyHasObj: 0,
      inputStage: {
        stage: 'IXSCAN',
        nReturned: 14104,
        executionTimeMillisEstimate: 1,
        works: 14105,
        advanced: 14104,
        needTime: 0,
        needYield: 0,
        saveState: 14,
        restoreState: 14,
        isEOF: 1,
        keyPattern: { top_critic_rating: 1 },
        indexName: 'top_critic_rating_1',
        isMultiKey: false,
        multiKeyPaths: { top_critic_rating: [] },
        isUnique: false,
        isSparse: false,
          isPartial: false,
          indexVersion: 2,
          direction: 'forward',
          indexBounds: { top_critic_rating: [ '[MinKey, MaxKey]' ] },
          keysExamined: 14104,
          seeks: 1,
          dupsTested: 0,
          dupsDropped: 0
        }
      }
    },
  command: {
    find: 'movie',
    filter: {},
    sort: { top_critic_rating: 1 },
    '$db': 'rottenMovies'
  },
  serverInfo: {
    host: 'Profile2022LARGE10',
    port: 27017,
    version: '6.0.3',
    gitVersion: 'f803681c3ae19817d31958965850193de067c516'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1673285423, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "
    hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1673285423, i: 1 })
}
```

Listing 15: Test

C .4 user rating

Before the index

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'rottenMovies.movie',
    indexFilterSet: false,
    parsedQuery: {},
    queryHash: '3E9B1E6C',
    planCacheKey: '3E9B1E6C',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'SORT',
      sortPattern: { user_rating: 1 },
      memLimit: 104857600,
      type: 'simple',
      inputStage: { stage: 'COLLSCAN', direction: 'forward' }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 14104,
    executionTimeMillis: 1779,
    totalKeysExamined: 0,
    totalDocsExamined: 14104,
    executionStages: {
      stage: 'SORT',
      nReturned: 14104,
      executionTimeMillisEstimate: 1698,
      works: 28211,
      advanced: 14104,
      needTime: 14106,
      needYield: 0,
      saveState: 45,
      restoreState: 45,
      isEOF: 1,
      sortPattern: { user_rating: 1 },
      memLimit: 104857600,
      type: 'simple',
      totalDataSizeSorted: 262640385,
      usedDisk: true,
      spills: 3,
      inputStage: {
        stage: 'COLLSCAN',
        nReturned: 14104,
        executionTimeMillisEstimate: 0,
        works: 14106,
        advanced: 14104,
        needTime: 1,
        needYield: 0,
        saveState: 45,
        restoreState: 45,
        isEOF: 1,
        direction: 'forward',
        docsExamined: 14104
      }
    }
  },
  command: {
    find: 'movie',
    filter: {},
    sort: { user_rating: 1 },
    '$db': 'rottenMovies'
  },
  serverInfo: {
    host: 'Profile2022LARGE10',
    port: 27017,
    version: '6.0.3',
    gitVersion: 'f803681c3ae19817d31958965850193de067c516'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1673287353, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "
  hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1673287353, i: 1 })
}
```

Listing 16: Test

After the index

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'rottenMovies.movie',
    indexFilterSet: false,
    parsedQuery: {},
    queryHash: '3E9B1E6C',
    planCacheKey: '3E9B1E6C',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { user_rating: 1 },
        indexName: 'user_rating_1',
        isMultiKey: false,
        multiKeyPaths: { user_rating: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { user_rating: [ '[MinKey, MaxKey]' ] }
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 14104,
    executionTimeMillis: 25,
    totalKeysExamined: 14104,
    totalDocsExamined: 14104,
    executionStages: {
      stage: 'FETCH',
      nReturned: 14104,
      executionTimeMillisEstimate: 5,
      works: 14105,
      advanced: 14104,
      needTime: 0,
      needYield: 0,
      saveState: 14,
      restoreState: 14,
      isEOF: 1,
      docsExamined: 14104,
      alreadyHasObj: 0,
      inputStage: {
        stage: 'IXSCAN',
        nReturned: 14104,
        executionTimeMillisEstimate: 2,
        works: 14105,
        advanced: 14104,
        needTime: 0,
        needYield: 0,
        saveState: 14,
        restoreState: 14,
        isEOF: 1,
        keyPattern: { user_rating: 1 },
        indexName: 'user_rating_1',
        isMultiKey: false,
        multiKeyPaths: { user_rating: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { user_rating: [ '[MinKey, MaxKey]' ] },
        keysExamined: 14104,
        seeks: 1,
        dupsTested: 0,
        dupsDropped: 0
      }
    }
  },
  command: {
    find: 'movie',
    filter: {},
    sort: { user_rating: 1 },
    '$db': 'rottenMovies'
  },
  serverInfo: {
    host: 'Profile2022LARGE10',
    port: 27017,
    version: '6.0.3',
    gitVersion: 'f803681c3ae19817d31958965850193de067c516'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1673285383, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "
    hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1673285383, i: 1 })
}
```

Listing 17: Test

C .5 personnel.primaryName

Before the index

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'rottenMovies.movie',
    indexFilterSet: false,
    parsedQuery: { 'personnel.primaryName': { '$eq': '' } },
    queryHash: 'E212F03B',
    planCacheKey: 'E212F03B',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'COLLSCAN',
      filter: { 'personnel.primaryName': { '$eq': '' } },
      direction: 'forward'
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 0,
    executionTimeMillis: 47,
    totalKeysExamined: 0,
    totalDocsExamined: 14104,
    executionStages: {
      stage: 'COLLSCAN',
      filter: { 'personnel.primaryName': { '$eq': '' } },
      nReturned: 0,
      executionTimeMillisEstimate: 9,
      works: 14106,
      advanced: 0,
      needTime: 14105,
      needYield: 0,
      saveState: 14,
      restoreState: 14,
      isEOF: 1,
      direction: 'forward',
      docsExamined: 14104
    }
  },
  command: {
    find: 'movie',
    filter: { 'personnel.primaryName': '' },
    '$db': 'rottenMovies'
  },
  serverInfo: {
    host: 'Profile2022LARGE10',
    port: 27017,
    version: '6.0.3',
    gitVersion: 'f803681c3ae19817d31958965850193de067c516'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1673287593, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "
    hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1673287593, i: 1 })
}
```

Listing 18: Test

After the index

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'rottenMovies.movie',
    indexFilterSet: false,
    parsedQuery: { 'personnel.primaryName': { '$eq': '' } },
    queryHash: 'E212F03B',
    planCacheKey: '9D4A6814',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { 'personnel.primaryName': 1 },
        indexName: 'personnel.primaryName_1',
        isMultiKey: true,
        multiKeyPaths: { 'personnel.primaryName': [ 'personnel' ] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { 'personnel.primaryName': [ '["", ""]' ] }
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 0,
    executionTimeMillis: 0,
    totalKeysExamined: 0,
    totalDocsExamined: 0,
    executionStages: {
      stage: 'FETCH',
      nReturned: 0,
      executionTimeMillisEstimate: 0,
      works: 1,
      advanced: 0,
      needTime: 0,
      needYield: 0,
      saveState: 0,
      restoreState: 0,
      isEOF: 1,
      docsExamined: 0,
      alreadyHasObj: 0,
      inputStage: {
        stage: 'IXSCAN',
        nReturned: 0,
        executionTimeMillisEstimate: 0,
        works: 1,
        advanced: 0,
        needTime: 0,
        needYield: 0,
        saveState: 0,
        restoreState: 0,
        isEOF: 1,
        keyPattern: { 'personnel.primaryName': 1 },
        indexName: 'personnel.primaryName_1',
        isMultiKey: true,
        multiKeyPaths: { 'personnel.primaryName': [ 'personnel' ] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { 'personnel.primaryName': [ '["", ""]' ] },
        keysExamined: 0,
        seeks: 1,
        dupsTested: 0,
        dupsDropped: 0
      }
    }
  },
  command: {
    find: 'movie',
    filter: { 'personnel.primaryName': '' },
    '$db': 'rottenMovies'
  },
  serverInfo: {
    host: 'Profile2022LARGE10',
    port: 27017,
    version: '6.0.3',
    gitVersion: 'f803681c3ae19817d31958965850193de067c516'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1673287763, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "
hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1673287763, i: 1 })
}
```

Listing 19: Test

D MongoDB indexes:User collection

D .1 username

Before the index

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'rottenMovies.user',
    indexFilterSet: false,
    parsedQuery: { username: { '$eq': 'Abbie Bernstein' } },
    queryHash: '7D9BB680',
    planCacheKey: '7D9BB680',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'COLLSCAN',
      filter: { username: { '$eq': 'Abbie Bernstein' } },
      direction: 'forward'
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 1,
    executionTimeMillis: 6,
    totalKeysExamined: 0,
    totalDocsExamined: 8339,
    executionStages: {
      stage: 'COLLSCAN',
      filter: { username: { '$eq': 'Abbie Bernstein' } },
      nReturned: 1,
        executionTimeMillisEstimate: 0,
        works: 8341,
        advanced: 1,
        needTime: 8339,
        needYield: 0,
        saveState: 8,
        restoreState: 8,
        isEOF: 1,
        direction: 'forward',
        docsExamined: 8339
      }
    },
    command: {
      find: 'user',
      filter: { username: 'Abbie Bernstein' },
      '$db': 'rottenMovies'
    },
    serverInfo: {
      host: 'Profile2022LARGE10',
      port: 27017,
      version: '6.0.3',
      gitVersion: 'f803681c3ae19817d31958965850193de067c516'
    },
    serverParameters: {
      internalQueryFacetBufferSizeBytes: 104857600,
      internalQueryFacetMaxOutputDocSizeBytes: 104857600,
      internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
      internalDocumentSourceGroupMaxMemoryBytes: 104857600,
      internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
      internalQueryProhibitBlockingMergeOnMongoS: 0,
      internalQueryMaxAddToSetBytes: 104857600,
      internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
    },
    ok: 1,
    '$clusterTime': {
      clusterTime: Timestamp({ t: 1673280753, i: 1 }),
      signature: {
        hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "
      hex"), 0),
        keyId: Long("0")
      }
    },
    operationTime: Timestamp({ t: 1673280753, i: 1 })
  }
```

Listing 20: Test

After the index

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'rottenMovies.user',
    indexFilterSet: false,
    parsedQuery: { username: { '$eq': 'Abbie Bernstein' } },
    queryHash: '7D9BB680',
    planCacheKey: '24069050',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { username: 1 },
        indexName: 'username_1',
        isMultiKey: false,
        multiKeyPaths: { username: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { username: [ '["Abbie Bernstein", "Abbie Bernstein"]' ] }
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 1,
    executionTimeMillis: 1,
    totalKeysExamined: 1,
    totalDocsExamined: 1,
    executionStages: {
      stage: 'FETCH',
      nReturned: 1,
      executionTimeMillisEstimate: 1,
      works: 2,
      advanced: 1,
      needTime: 0,
      needYield: 0,
      saveState: 0,
      restoreState: 0,
      isEOF: 1,
      docsExamined: 1,
      alreadyHasObj: 0,
      inputStage: {
        stage: 'IXSCAN',
        nReturned: 1,
        executionTimeMillisEstimate: 1,
        works: 2,
        advanced: 1,
        needTime: 0,
        needYield: 0,
        saveState: 0,
        restoreState: 0,
        isEOF: 1,
        keyPattern: { username: 1 },
        indexName: 'username_1',
        isMultiKey: false,
        multiKeyPaths: { username: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { username: [ '["Abbie Bernstein", "Abbie Bernstein"]' ] },
        keysExamined: 1,
        seeks: 1,
        dupsTested: 0,
          dupsDropped: 0
        }
      }
    },
    command: {
      find: 'user',
      filter: { username: 'Abbie Bernstein' },
      '$db': 'rottenMovies'
    },
    serverInfo: {
      host: 'Profile2022LARGE10',
      port: 27017,
      version: '6.0.3',
      gitVersion: 'f803681c3ae19817d31958965850193de067c516'
    },
    serverParameters: {
      internalQueryFacetBufferSizeBytes: 104857600,
      internalQueryFacetMaxOutputDocSizeBytes: 104857600,
      internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
      internalDocumentSourceGroupMaxMemoryBytes: 104857600,
      internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
      internalQueryProhibitBlockingMergeOnMongoS: 0,
      internalQueryMaxAddToSetBytes: 104857600,
      internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
    },
    ok: 1,
    '$clusterTime': {
      clusterTime: Timestamp({ t: 1673285013, i: 1 }),
      signature: {
        hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "
  hex"), 0),
        keyId: Long("0")
      }
    },
    operationTime: Timestamp({ t: 1673285013, i: 1 })
}
```

Listing 21: Test

D .2 date of birth

Before the index

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'rottenMovies.user',
    indexFilterSet: false,
    parsedQuery: {
      date_of_birth: {
        '$eq': 'Mon Jan 09 2023 17:05:07 GMT+0000 (Western European Standard
  Time)'
      }
    },
    queryHash: 'D7A0117C',
    planCacheKey: 'D7A0117C',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
        stage: 'COLLSCAN',
        filter: {
          date_of_birth: {
            '$eq': 'Mon Jan 09 2023 17:05:07 GMT+0000 (Western European
    Standard Time)'
          }
        },
        direction: 'forward'
      },
      rejectedPlans: []
    },
    executionStats: {
      executionSuccess: true,
      nReturned: 0,
      executionTimeMillis: 32,
      totalKeysExamined: 0,
      totalDocsExamined: 8339,
      executionStages: {
        stage: 'COLLSCAN',
        filter: {
          date_of_birth: {
            '$eq': 'Mon Jan 09 2023 17:05:07 GMT+0000 (Western European
    Standard Time)'
          }
        },
        nReturned: 0,
        executionTimeMillisEstimate: 23,
        works: 8341,
        advanced: 0,
        needTime: 8340,
        needYield: 0,
        saveState: 8,
        restoreState: 8,
        isEOF: 1,
        direction: 'forward',
        docsExamined: 8339
      }
    },
    command: {
      find: 'user',
      filter: {
        date_of_birth: 'Mon Jan 09 2023 17:05:07 GMT+0000 (Western European
    Standard Time)'
      },
      '$db': 'rottenMovies'
    },
    serverInfo: {
      host: 'Profile2022LARGE10',
      port: 27017,
      version: '6.0.3',
      gitVersion: 'f803681c3ae19817d31958965850193de067c516'
    },
    serverParameters: {
      internalQueryFacetBufferSizeBytes: 104857600,
      internalQueryFacetMaxOutputDocSizeBytes: 104857600,
      internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
      internalDocumentSourceGroupMaxMemoryBytes: 104857600,
      internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
      internalQueryProhibitBlockingMergeOnMongoS: 0,
      internalQueryMaxAddToSetBytes: 104857600,
      internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
```

```
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1673283903, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "
hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1673283903, i: 1 })
}
```

Listing 22: Test

After the index

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'rottenMovies.user',
    indexFilterSet: false,
    parsedQuery: {
      date_of_birth: {
        '$eq': 'Mon Jan 09 2023 17:24:42 GMT+0000 (Western European Standard
Time)'
      }
    },
    queryHash: 'D7A0117C',
    planCacheKey: '90F68BB6',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { date_of_birth: 1 },
        indexName: 'date_of_birth_1',
        isMultiKey: false,
        multiKeyPaths: { date_of_birth: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: {
          date_of_birth: [
            '["Mon Jan 09 2023 17:24:42 GMT+0000 (Western European Standard
Time)", "Mon Jan 09 2023 17:24:42 GMT+0000 (Western European Standard
Time)"]'
          ]
        }
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 0,
    executionTimeMillis: 1,
    totalKeysExamined: 0,
    totalDocsExamined: 0,
    executionStages: {
      stage: 'FETCH',
      nReturned: 0,
      executionTimeMillisEstimate: 0,
      works: 1,
      advanced: 0,
      needTime: 0,
      needYield: 0,
      saveState: 0,
      restoreState: 0,
      isEOF: 1,
      docsExamined: 0,
      alreadyHasObj: 0,
      inputStage: {
        stage: 'IXSCAN',
        nReturned: 0,
        executionTimeMillisEstimate: 0,
        works: 1,
        advanced: 0,
        needTime: 0,
        needYield: 0,
        saveState: 0,
        restoreState: 0,
        isEOF: 1,
        keyPattern: { date_of_birth: 1 },
        indexName: 'date_of_birth_1',
        isMultiKey: false,
        multiKeyPaths: { date_of_birth: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: {
          date_of_birth: [
            '["Mon Jan 09 2023 17:24:42 GMT+0000 (Western European Standard
  Time)", "Mon Jan 09 2023 17:24:42 GMT+0000 (Western European Standard
  Time)"]'
          ]
        },
        keysExamined: 0,
        seeks: 1,
        dupsTested: 0,
        dupsDropped: 0
      }
    }
  },
  command: {
    find: 'user',
    filter: {
      date_of_birth: 'Mon Jan 09 2023 17:24:42 GMT+0000 (Western European
  Standard Time)'
    },
    '$db': 'rottenMovies'
  },
  serverInfo: {
    host: 'Profile2022LARGE10',
    port: 27017,
    version: '6.0.3',
    gitVersion: 'f803681c3ae19817d31958965850193de067c516'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1673285073, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "
   hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1673285073, i: 1 })
}
```

Listing 23: Test

E Application code

E .1 Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.
   org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache
   .org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>3.0.0</version>
      <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>it.unipi.dii.lsmsdb</groupId>
   <artifactId>rottenMovies</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>rottenMovies</name>
   <description>Project for the rotten movies service</description>
   <properties>
      <java.version>19</java.version>
   </properties>
   <dependencies>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-thymeleaf</artifactId>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-web</artifactId>
      </dependency>
```

```xml
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-mongodb-reactive</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-neo4j</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>io.projectreactor</groupId>
            <artifactId>reactor-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.10</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-annotations</artifactId>
            <version>2.14.1</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.14.0</version>
        </dependency>
        <dependency>
            <groupId>org.neo4j.driver</groupId>
            <artifactId>neo4j-java-driver</artifactId>
            <version>5.3.0</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Listing 24: Test