

## A Python Code

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

```
1
2 import pandas as pd
3 from google.colab import drive
4 drive.mount('/content/drive')
5 numberofrows=None #100000
6 title_basics = pd.read_csv("/content/drive/MyDrive/Dataset/Original/
    title_basics.tsv",sep='\t',rows=numberofrows,header=0)
7 title_principals = pd.read_csv("/content/drive/MyDrive/Dataset/Original/
    title_principals.tsv",rows=numberofrows,sep='\t',header=0)
8
9 keep_col = ["tconst","titleType","primaryTitle","originalTitle","startYear","
    runtimeMinutes","genres"]
10 title_basics = title_basics[keep_col]
11 title_basics = title_basics[title_basics["titleType"].str.contains("movie")
    == True]
12
13 print(title_basics.head(3))
14
15 merged1=pd.merge(title_basics ,title_principals ,how='inner',on='tconst')
16 del title_basics ,title_principals
17 print(merged1)
18
19 name_basics=pd.read_csv("/content/drive/MyDrive/Dataset/Original/name_basics.
    tsv",sep='\t',rows=numberofrows,header=0)
20
21 merged2=pd.merge(merged1 ,name_basics ,how='inner',on='nconst')
22 del merged1
23 merged2=merged2.drop(columns=["ordering","nconst","birthYear","deathYear","
    knownForTitles","primaryProfession"])
24 del name_basics
25 print(merged2)
26
27 category=merged2.groupby('tconst')['category'].apply(list).reset_index(name='
    category')
28 job=merged2.groupby('tconst')['job'].apply(list).reset_index(name='job')
29 characters=merged2.groupby('tconst')['characters'].apply(list).reset_index(
    name='characters')
30 primaryName=merged2.groupby('tconst')['primaryName'].apply(list).reset_index(
    name='primaryName')
31 result=merged2.drop_duplicates(subset=['tconst'])
32 result=result.drop(['category',axis=1).drop(['job',axis=1).drop(['
    characters',axis=1).drop(['primaryName',axis=1)
33 result=result.merge(category,on='tconst').merge(job,on='tconst').merge(
    characters,on='tconst').merge(primaryName,on='tconst')
34 print(result)
35
36 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

```

2 import pandas as pd
3 from google.colab import drive
4 drive.mount('/content/drive')
5 numberofrows=None #100000
6 movies = pd.read_csv("/content/drive/MyDrive/Dataset/Original/rotten_movies.
    csv",nrows=numberofrows,header=0)
7 reviews = pd.read_csv("/content/drive/MyDrive/Dataset/Original/rotten_reviews
    .csv",nrows=numberofrows,header=0)
8 to_keep = ["rotten_tomatoes_link", "movie_title", "production_company", "
    critics_consensus",
9            "tomatometer_status", "tomatometer_rating", "tomatometer_count",
10           "audience_status", "audience_rating", "audience_count",
11           "tomatometer_top_critics_count", "tomatometer_fresh_critics_count"
12           ,
13           "tomatometer_rotten_critics_count"]
14 movies = movies[to_keep]
15 to_drop = ["publisher_name"]
16 reviews=reviews.drop(columns=to_drop)
17
18 merged=pd.merge(movies, reviews ,how='inner',on="rotten_tomatoes_link")
19 print(merged)
20
21 categories = {}
22 arr = ["critic_name", "top_critic", "review_type", "review_score", "
    review_date", "review_content"]
23 for x in arr:
24     categories[x]=merged.groupby('rotten_tomatoes_link')[x].apply(list).
    reset_index(name=x)
25
26 result=merged.drop_duplicates(subset=['rotten_tomatoes_link'])
27 for x in arr:
28     result=result.drop([x], axis=1)
29 for x in arr:
30     result=result.merge(categories[x],on='rotten_tomatoes_link')
31
32 print(result)
33
34 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

```

1
2 import pandas as pd
3 from google.colab import drive
4 drive.mount('/content/drive')
5 numberofrows=None
6 imdb = pd.read_csv("/content/drive/MyDrive/Dataset/resultSetFinale.csv",nrows
    =numberofrows,header=0)
7 rotten = pd.read_csv("/content/drive/MyDrive/Dataset/resultSetRotten.csv",
    nrows=numberofrows,header=0)
8
9 merged = {}
10 choose = ['primaryTitle', 'originalTitle']
11 rowHeadDataset = 20
12 for x in choose:

```

```

13 merged[x]=pd.merge(imdb,rotten,how='inner', left_on=x, right_on='
    movie_title')
14 print(x)
15 merged[x]=merged[x].drop_duplicates(subset=[x])
16 merged[x]=merged[x].drop(columns=['titleType','tomatometer_count','
    tomatometer_top_critics_count'])
17 merged[x]=merged[x].rename(columns={'startYear':'year'})
18 merged[x]=merged[x].drop(columns=['rotten_tomatoes_link','movie_title']+
    [j for j in choose if j!=x])
19 print(len(pd.unique(merged[x][x])))
20 print(list(merged[x]))
21 print("=====")
22 merged[x].to_csv(f"/content/drive/MyDrive/Dataset/ImdbJoinRotten{x}.csv",
    index=False)
23 merged[x]=merged[x].head(rowHeadDataset)
24 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

```

1
2 import pandas as pd
3 from ast import literal_eval
4 from google.colab import drive
5 drive.mount('/content/drive')
6
7 numberofrows=None
8 df = pd.read_csv("/content/drive/MyDrive/Dataset/ImdbJoinRottenprimaryTitle.
    csv",nrows=numberofrows,header=0)
9
10 #print([x.split(',') for x in df['genres']])
11 print(df)
12
13 col = ["primaryName","category","job","characters"]
14 col1 = ["critic_name","top_critic","review_type","review_score","review_date"
    , "review_content"]
15
16 df['personnel'] = ""
17 df['review'] = ""
18
19 for row in range(df[col[0]].size):
20     it = df['genres'][row]
21     df['genres'][row] = ['"'+x+'" for x in it.split(',') if it != '\\N'
        else []]
22     tmp = []
23     for c in col:
24         tmp.append({c:eval(df[c][row])})
25     res = []
26     for c in range(len(tmp[0][col[0]])):
27         res.append({})
28     for i, j in zip(col, tmp):
29         for idx, x in enumerate(j[i]):
30             #print(i, idx, x)
31             if x != '\\N':
32                 if i == 'characters':
33                     x = eval(x)

```

```

34     res[idx][ "'" + i + "'" ] = "'" + str(x).replace("'", "##single-quote##")
35     ").replace('"', "##double-quote##") + "'"
36 df['personnel'][row] = list(res)
37 #print(res)
38 ###
39 tmp = []
40 for c in col1:
41     to_eval = df[c][row].replace('nan', 'None')
42     arr = eval(to_eval)
43     if c == "review_date":
44         for i, elem in enumerate(arr):
45             arr[i] = elem + "T00:00:00.000+00:00"
46     tmp.append({c: arr})
47 #print(tmp)
48 res = []
49 for c in range(len(tmp[0][col1[0]])):
50     res.append({})
51 for i, j in zip(col1, tmp):
52     for idx, x in enumerate(j[i]):
53         #print(i, idx, x)
54         if x != '\\N':
55             res[idx][ "'" + i + "'" ] = "'" + str(x).replace("True", "true").
56             replace("False", "false").replace("'", "##single-quote##").replace('"', "
57             ##double-quote##") + "'"
58 df['review'][row] = list(res)
59 #df['review'][row] = eval(str(res))
60 #print(res)
61 #print()
62 df=df.drop(columns=col)
63 df=df.drop(columns=col1)
64 df=df.drop(columns=['tconst'])
65
66 print(df["review"][0])
67
68 it = df['personnel'][0][4]['review_content']
69 print(type(it))
70 print(it)
71
72 df.to_csv("/content/drive/MyDrive/Dataset/
73     movieCollectionEmbeddedReviewPersonnel.csv", index=False)
74 df = df.head(20)
75 df.to_csv("/content/drive/MyDrive/Dataset/
76     headmovieCollectionEmbeddedReviewPersonnel.csv", index=False)

```

Listing 4: Test

## A .5 Generates a hashed password for all the users

```

1 import hashlib
2 #from pprint import pprint as print
3 from pymongo import MongoClient
4
5 def get_database():
6     CONNECTION_STRING = "mongodb://localhost:27017"
7     client = MongoClient(CONNECTION_STRING)
8     return client['rottenMovies']
9
10 if __name__ == "__main__":

```

```

11     dbname = get_database()
12     collection = dbname[ 'user' ]
13     total = collection.count_documents({})
14     for i, user in enumerate(collection.find()):
15         all_reviews = user[ 'last_3_reviews' ]
16         sorted_list = sorted(all_reviews, key=lambda t: t[ 'review_date' ])
17         [-3:]
18         hashed = hashlib.md5(user[ "username" ].encode()).hexdigest()
19
20         newvalues = { "$set": { 'password': hashed, 'last_3_reviews':
sorted_list } }
21         filter = { 'username': user[ 'username' ] }
22         collection.update_one(filter, newvalues)
23         print(f"{i/total:%}\r", end='')
24     print()

```

Listing 5: Test

## A .6 Generates the graph database

```

1  from pymongo import MongoClient
2  from neo4j import GraphDatabase
3  from random import randint, shuffle
4
5  def get_database():
6      CONNECTION_STRING = "mongodb://localhost:27017"
7      client = MongoClient(CONNECTION_STRING)
8      return client[ 'rottenMovies' ]
9
10 class Neo4jGraph:
11
12     def __init__(self, uri, user, password):
13         self.driver = GraphDatabase.driver(uri, auth=(user, password),
database="rottenmoviesgraphdb")
14
15     def close(self):
16         self.driver.close()
17
18     def addUser(self, uid, name, isTop):
19         with self.driver.session() as session:
20             if isTop:
21                 result = session.execute_write(self._addTopCritic, uid, name)
22             else:
23                 result = session.execute_write(self._addUser, uid, name)
24
25     def addMovie(self, mid, title):
26         with self.driver.session() as session:
27             result = session.execute_write(self._addMovie, mid, title)
28
29     def addReview(self, name, mid, freshness, content, date):
30         with self.driver.session() as session:
31             result = session.execute_write(self._addReview, name, mid,
freshness, content, date)
32
33     def addFollow(self, uid, cid):
34         with self.driver.session() as session:
35             result = session.execute_write(self._addFollow, uid, cid)
36

```

```

37     @staticmethod
38     def _addUser(tx, uid, name):
39         query = "CREATE (n:User{id:\\"" + str(uid) + "\", name:\\"" + name.
replace("'", '\\') + "\"})"
40         #print(query)
41         result = tx.run(query)
42
43     @staticmethod
44     def _addTopCritic(tx, cid, name):
45         query = "CREATE(m:TopCritic{id:\\"" + str(cid) + "\", name:\\"" + name.
replace("'", '\\') + "\"})"
46         #print(query)
47         result = tx.run(query)
48
49     @staticmethod
50     def _addMovie(tx, mid, title):
51         query = "CREATE(o:Movie{id:\\"" + str(mid) + "\", title:\\"" + title.
replace("'", '\\') + "\"})"
52         #print(query)
53         result = tx.run(query)
54
55     @staticmethod
56     def _addReview(tx, name, mid, freshness, content, date): # date in format
YYYY-mm-dd, freshness in [TRUE, FALSE]
57         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)"
58         #print(query)
59         result = tx.run(query)
60
61     @staticmethod
62     def _addFollow(tx, uid, cid):
63         query = "MATCH(n:User{id:\\"" + str(uid) + "\"}), (m:TopCritic{id:\\""
+ str(cid) + "\"}) CREATE (n)-[r:FOLLOWS]->(m)"
64         #print(query)
65         result = tx.run(query)
66
67 if __name__ == "__main__":
68     # dbs initialization
69     dbname = get_database()
70     graphDB = Neo4jGraph("bolt://localhost:7687", "neo4j", "password")
71
72     # user creation
73     collection = dbname['user']
74     total = collection.count_documents({})
75     print(f"user {total} = ")
76     for i, user in enumerate(list(collection.find({}, {"_id":1, "username":1,
"date_of_birth":1}))) :
77         graphDB.addUser(user['_id'], user['username'], 'date_of_birth' not in
user)
78         if not i%100:
79             print(f" {(i+1)/total:%}\r", end='')
80
81     # movie creation and review linking
82     collection = dbname['movie']
83     total = collection.count_documents({})
84     print(f"\nmovie {total} = ")
85     for i, movie in enumerate(list(collection.find({}, {"_id":1, "
primaryTitle":1, "review":1}))) :
86         graphDB.addMovie(movie['_id'], movie['primaryTitle'])

```

```

87     movie['review'] = list({v['critic_name']:v for v in movie['review']}.
values()) # make unique reviews per critic
88     for rev in movie['review']:
89         graphDB.addReview(rev['critic_name'], movie['_id'], {"Fresh":
TRUE", "Rotten":"FALSE"}[rev['review_type']], str(rev['review_content'])
[:15], str(rev['review_date'])[:10])
90         print(f" {(i+1)/total:%}\r", end='')
91
92     # follow linking
93     collection = dbname['user']
94     uids = [x['_id'] for x in list(collection.find({"date_of_birth":{"$exists
":True}}, {"_id":1}))]
95     cids = [x['_id'] for x in list(collection.find({"date_of_birth":{"$exists
":False}}, {"_id":1}))]
96     total = len(uids)
97     print(f"\nfollow {total = }")
98     for i, user in enumerate(uids):
99         shuffle(cids)
100         for j in range(randint(0, 20)):
101             graphDB.addFollow(user, cids[j])
102             print(f" {i/total:%}\r", end='')
103
104     graphDB.close()

```

Listing 6: Test

## B Mongosh scripts

### B.1 Perform the escape on the string fields

```
1
2 db.movie.find().forEach(
3   x => {
4     print(x.primaryTitle);
5     x.review = JSON.parse(
6       x.review.replaceAll('"\'', '"')
7         .replaceAll('\\"', '"')
8         .replaceAll('false', 'false')
9         .replaceAll('true', 'true')
10        .replaceAll('None', 'null')
11        .replaceAll(/\\x\d{2}/g, "")
12        .replaceAll("##single-quote##", '\')
13        .replaceAll("##double-quote##", '\\')
14        .replaceAll("\\x", "x")
15    );
16    x.personnel = JSON.parse(
17      x.personnel.replaceAll('"\'', '"')
18        .replaceAll('\\"', '"')
19        .replaceAll('None', 'null')
20        .replaceAll("##single-quote##", '\')
21        .replaceAll("##double-quote##", '\\')
22        .replaceAll('["', '[')
23        .replaceAll('"\\"', '"')
24        .replaceAll('\'', ']')
25        .replaceAll('\\\""', '"')
26        .replaceAll(/(\[[^\]]*\)\\"/, '\\'([^\]]*\))/g, '$1', '$2')
27        .replaceAll(/(\[[^\]]*\)\\'/, '\\'([^\]]*\))/g, '$1', '$2')
28        .replaceAll(/(\[[^\]]*\)\\"/, '\\'([^\]]*\))/g, '$1', '$2')
29    );
30    x.genres = JSON.parse(
31      x.genres.replaceAll('"\'', '"')
32        .replaceAll('\\"', '"')
33        .replaceAll('None', 'null')
34        .replaceAll("##single-quote##", '\')
35        .replaceAll("##double-quote##", '\\')
36    );
37    db.movie.updateOne(
38      {"_id": x._id},
39      {$set:
40        {
41          "review": x.review,
42          "personnel": x.personnel,
43          "genres": x.genres,
44          "runtimeMinutes": parseInt(x.runtimeMinutes),
45          "year": parseInt(x.year),
46          "tomatometer_rating": parseFloat(x.tomatometer_rating),
47          "audience_rating": parseFloat(x.audience_rating),
48          "audience_count": parseFloat(x.audience_count),
49          "tomatometer_fresh_critics_count": parseInt(x.
50tomatometer_fresh_critics_count),
51          "tomatometer_rotten_critics_count": parseInt(x.
52tomatometer_rotten_critics_count)
53        }
54      }
55    );
```



```

54     }
55 );

```

Listing 7: Test

## B .2 Normalize the date field in the DB

```

1 total = db.movie.find().count();
2 i = 0;
3 db.movie.find().forEach(
4     x => {
5         print(x.primaryTitle);
6         x.review.forEach(rev =>{
7             if(typeof (rev.review_date) === "string" ){
8                 db.movie.updateOne(
9                     {primaryTitle: x.primaryTitle },
10                    { $set: { "review.$[elem].review_date" : new Date(rev.
11                        review_date) } },
12                    { arrayFilters: [ { "elem.critic_name": rev.critic_name }
13                        ] }
14                )
15            }
16        })
17        print(100*i++/total);
18    });

```

Listing 8: Test

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

```

1 total = db.runCommand({ distinct: "movie", key: "review.critic_name", query:
2     {"review.critic_name":{"$ne: null}}}).values.length
3 i = 0;
4 db.runCommand(
5     { distinct: "movie", key: "review.critic_name", query: {"review.critic_name"
6         :{"$ne: null"}}}).values.forEach(
7     (x) => {
8         review_arr = []
9         movie_arr = []
10        is_top = false
11        db.movie.aggregate(
12            [
13                { $project:
14                    {
15                        index: { $indexOfArray: ["$review.critic_name", x] },
16                        primaryTitle: 1
17                    }
18                },
19                { $match: { index: { $gt: -1 } } }
20            ]
21        ).forEach(
22            y => {
23                tmp = db.movie.aggregate([
24                    {
25                        $project:
26                        {
27                            top_critic: {

```

```

25         $arrayElemAt: [ "$review.top_critic", y.index ]
26     },
27     primaryTitle: y.primaryTitle,
28     review_type: {
29         $arrayElemAt: [ "$review.review_type", y.index
30     ]
31     },
32     review_score: {
33         $arrayElemAt: [ "$review.review_score", y.
34     index ]
35     },
36     review_date: {
37         $arrayElemAt: [ "$review.review_date", y.index
38     ]
39     },
40     review_content: {
41         $arrayElemAt: [ "$review.review_content", y.
42     index ]
43     }
44     },
45     $match: { _id: { $eq: y._id } }
46 }
47 ).toArray() [0];
48 is_top |= tmp.top_critic;
49 review_arr.push(tmp)
50 //movie_arr.push(tmp._id)
51 movie_arr.push({"movie_id": tmp._id, "primaryTitle": y.
52 primaryTitle, "review_index": y.index})
53 })
54
55 name_parts = x.split(/\s/)
56 first_name = name_parts.splice(0, 1)[0]
57 last_name = name_parts.join(' ')
58
59 print(100*i++/total, x, is_top)
60 //print(first_name, ': ', last_name)
61 //print(review_arr)
62 //print(movie_arr)
63 db.user.insertOne(
64 {
65     "username": x,
66     "password": "",
67     "first_name": first_name,
68     "last_name": last_name,
69     "registration_date": new Date("2000-01-01"),
70     "last_3_reviews": review_arr,
71     "reviews" : movie_arr
72 }
73 );
74 if (!is_top){
75     db.user.updateOne(
76     { "username": x },
77     { $set:
78     { "date_of_birth": new Date("1970-07-20") } }
79     )
80 }
81 print("=====")
82 }

```

## C MongoDB indexes:Movie collection

### C.1 primaryTitle

Before the index

```

1 {
2   explainVersion: '1',
3   queryPlanner: {
4     namespace: 'rottenMovies.movie',
5     indexFilterSet: false,
6     parsedQuery: { primaryTitle: { '$eq': 'Evidence' } },
7     queryHash: '9839850C',
8     planCacheKey: '9839850C',
9     maxIndexedOrSolutionsReached: false,
10    maxIndexedAndSolutionsReached: false,
11    maxScansToExplodeReached: false,
12    winningPlan: {
13      stage: 'COLLSCAN',
14      filter: { primaryTitle: { '$eq': 'Evidence' } },
15      direction: 'forward'
16    },
17    rejectedPlans: []
18  },
19  executionStats: {
20    executionSuccess: true,
21    nReturned: 1,
22    executionTimeMillis: 275,
23    totalKeysExamined: 0,
24    totalDocsExamined: 14104,
25    executionStages: {
26      stage: 'COLLSCAN',
27      filter: { primaryTitle: { '$eq': 'Evidence' } },
28      nReturned: 1,
29      executionTimeMillisEstimate: 245,
30      works: 14106,
31      advanced: 1,
32      needTime: 14104,
33      needYield: 0,
34      saveState: 18,
35      restoreState: 18,
36      isEOF: 1,
37      direction: 'forward',
38      docsExamined: 14104
39    }
40  },
41  command: {
42    find: 'movie',
43    filter: { primaryTitle: 'Evidence' },
44    '$db': 'rottenMovies'
45  },
46  serverInfo: {
47    host: 'Profile2022LARGE10',
48    port: 27017,
49    version: '6.0.3',

```

```

50   gitVersion: 'f803681c3ae19817d31958965850193de067c516'
51 },
52 serverParameters: {
53   internalQueryFacetBufferSizeBytes: 104857600,
54   internalQueryFacetMaxOutputDocSizeBytes: 104857600,
55   internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
56   internalDocumentSourceGroupMaxMemoryBytes: 104857600,
57   internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
58   internalQueryProhibitBlockingMergeOnMongoS: 0,
59   internalQueryMaxAddToSetBytes: 104857600,
60   internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
61 },
62 ok: 1,
63 '$clusterTime': {
64   clusterTime: Timestamp({ t: 1673280853, i: 1 }),
65   signature: {
66     hash: Binary(Buffer.from("0000000000000000000000000000000000", "
67     hex"), 0),
68     keyId: Long("0")
69   }
70 },
71 operationTime: Timestamp({ t: 1673280853, i: 1 })

```

Listing 10: Test

After the index

```

1 {
2   explainVersion: '1',
3   queryPlanner: {
4     namespace: 'rottenMovies.movie',
5     indexFilterSet: false,
6     parsedQuery: { primaryTitle: { '$eq': 'Evidence' } },
7     queryHash: '9839850C',
8     planCacheKey: 'B734708E',
9     maxIndexedOrSolutionsReached: false,
10    maxIndexedAndSolutionsReached: false,
11    maxScansToExplodeReached: false,
12    winningPlan: {
13      stage: 'FETCH',
14      inputStage: {
15        stage: 'IXSCAN',
16        keyPattern: { primaryTitle: 1 },
17        indexName: 'primaryTitle_1',
18        isMultiKey: false,
19        multiKeyPaths: { primaryTitle: [] },
20        isUnique: false,
21        isSparse: false,
22        isPartial: false,
23        indexVersion: 2,
24        direction: 'forward',
25        indexBounds: { primaryTitle: [ '['Evidence', 'Evidence']' ] }
26      }
27    },
28    rejectedPlans: []
29  },
30  executionStats: {
31    executionSuccess: true,
32    nReturned: 1,
33    executionTimeMillis: 1,
34    totalKeysExamined: 1,

```

```

35     totalDocsExamined: 1,
36     executionStages: {
37         stage: 'FETCH',
38         nReturned: 1,
39         executionTimeMillisEstimate: 0,
40         works: 2,
41         advanced: 1,
42         needTime: 0,
43         needYield: 0,
44         saveState: 0,
45         restoreState: 0,
46         isEOF: 1,
47         docsExamined: 1,
48         alreadyHasObj: 0,
49         inputStage: {
50             stage: 'IXSCAN',
51             nReturned: 1,
52             executionTimeMillisEstimate: 0,
53             works: 2,
54             advanced: 1,
55             needTime: 0,
56             needYield: 0,
57             saveState: 0,
58             restoreState: 0,
59             isEOF: 1,
60             keyPattern: { primaryTitle: 1 },
61             indexName: 'primaryTitle_1',
62             isMultiKey: false,
63             multiKeyPaths: { primaryTitle: [] },
64             isUnique: false,
65             isSparse: false,
66             isPartial: false,
67             indexVersion: 2,
68             direction: 'forward',
69             indexBounds: { primaryTitle: [ '["Evidence", "Evidence"]' ] },
70             keysExamined: 1,
71             seeks: 1,
72             dupsTested: 0,
73             dupsDropped: 0
74         }
75     },
76 },
77 command: {
78     find: 'movie',
79     filter: { primaryTitle: 'Evidence' },
80     '$db': 'rottenMovies'
81 },
82 serverInfo: {
83     host: 'Profile2022LARGE10',
84     port: 27017,
85     version: '6.0.3',
86     gitVersion: 'f803681c3ae19817d31958965850193de067c516'
87 },
88 serverParameters: {
89     internalQueryFacetBufferSizeBytes: 104857600,
90     internalQueryFacetMaxOutputDocSizeBytes: 104857600,
91     internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
92     internalDocumentSourceGroupMaxMemoryBytes: 104857600,
93     internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
94     internalQueryProhibitBlockingMergeOnMongoS: 0,
95     internalQueryMaxAddToSetBytes: 104857600,

```

```

96     internalDocumentSourceSet WindowFieldsMaxMemoryBytes: 104857600
97 },
98 ok: 1,
99 '$clusterTime': {
100     clusterTime: Timestamp({ t: 1673285103, i: 1 }),
101     signature: {
102         hash: Binary(Buffer.from("00000000000000000000000000000000", "
103         hex"), 0),
104         keyId: Long("0")
105     }
106 },
107 operationTime: Timestamp({ t: 1673285103, i: 1 })
108 }

```

Listing 11: Test

## C.2 year

Before the index

```

1 {
2   explainVersion: '1',
3   queryPlanner: {
4     namespace: 'rottenMovies.movie',
5     indexFilterSet: false,
6     parsedQuery: { year: { '$eq': 2012 } } },
7     queryHash: '412E8B51',
8     planCacheKey: '412E8B51',
9     maxIndexedOrSolutionsReached: false,
10    maxIndexedAndSolutionsReached: false,
11    maxScansToExplodeReached: false,
12    winningPlan: {
13      stage: 'COLLSCAN',
14      filter: { year: { '$eq': 2012 } } },
15      direction: 'forward'
16    },
17    rejectedPlans: []
18  },
19  executionStats: {
20    executionSuccess: true,
21    nReturned: 480,
22    executionTimeMillis: 13,
23    totalKeysExamined: 0,
24    totalDocsExamined: 14104,
25    executionStages: {
26      stage: 'COLLSCAN',
27      filter: { year: { '$eq': 2012 } } },
28      nReturned: 480,
29      executionTimeMillisEstimate: 1,
30      works: 14106,
31      advanced: 480,
32      needTime: 13625,
33      needYield: 0,
34      saveState: 14,
35      restoreState: 14,
36      isEOF: 1,
37      direction: 'forward',
38      docsExamined: 14104
39    }
40  },

```

```

41 command: { find: 'movie', filter: { year: 2012 }, '$db': 'rottenMovies' },
42 serverInfo: {
43   host: 'Profile2022LARGE10',
44   port: 27017,
45   version: '6.0.3',
46   gitVersion: 'f803681c3ae19817d31958965850193de067c516'
47 },
48 serverParameters: {
49   internalQueryFacetBufferSizeBytes: 104857600,
50   internalQueryFacetMaxOutputDocSizeBytes: 104857600,
51   internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
52   internalDocumentSourceGroupMaxMemoryBytes: 104857600,
53   internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
54   internalQueryProhibitBlockingMergeOnMongoS: 0,
55   internalQueryMaxAddToSetBytes: 104857600,
56   internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
57 },
58 ok: 1,
59 '$clusterTime': {
60   clusterTime: Timestamp({ t: 1673280923, i: 1 }),
61   signature: {
62     hash: Binary(Buffer.from("000000000000000000000000000000000000",
63     hex"), 0),
64     keyId: Long("0")
65   }
66 },
67 operationTime: Timestamp({ t: 1673280923, i: 1 })

```

Listing 12: Test

After the index

```

1 {
2   explainVersion: '1',
3   queryPlanner: {
4     namespace: 'rottenMovies.movie',
5     indexFilterSet: false,
6     parsedQuery: { year: { '$eq': 2012 } },
7     queryHash: '412E8B51',
8     planCacheKey: '62915BA3',
9     maxIndexedOrSolutionsReached: false,
10    maxIndexedAndSolutionsReached: false,
11    maxScansToExplodeReached: false,
12    winningPlan: {
13      stage: 'FETCH',
14      inputStage: {
15        stage: 'IXSCAN',
16        keyPattern: { year: 1 },
17        indexName: 'year_1',
18        isMultiKey: false,
19        multiKeyPaths: { year: [] },
20        isUnique: false,
21        isSparse: false,
22        isPartial: false,
23        indexVersion: 2,
24        direction: 'forward',
25        indexBounds: { year: [ '[2012, 2012]' ] }
26      }
27    },
28    rejectedPlans: []
29  },

```

```

30  executionStats: {
31    executionSuccess: true,
32    nReturned: 480,
33    executionTimeMillis: 2,
34    totalKeysExamined: 480,
35    totalDocsExamined: 480,
36    executionStages: {
37      stage: 'FETCH',
38      nReturned: 480,
39      executionTimeMillisEstimate: 0,
40      works: 481,
41      advanced: 480,
42      needTime: 0,
43      needYield: 0,
44      saveState: 0,
45      restoreState: 0,
46      isEOF: 1,
47      docsExamined: 480,
48      alreadyHasObj: 0,
49      inputStage: {
50        stage: 'IXSCAN',
51        nReturned: 480,
52        executionTimeMillisEstimate: 0,
53        works: 481,
54        advanced: 480,
55        needTime: 0,
56        needYield: 0,
57        saveState: 0,
58        restoreState: 0,
59        isEOF: 1,
60        keyPattern: { year: 1 },
61        indexName: 'year_1',
62        isMultiKey: false,
63        multiKeyPaths: { year: [] },
64        isUnique: false,
65        isSparse: false,
66        isPartial: false,
67        indexVersion: 2,
68        direction: 'forward',
69        indexBounds: { year: [ '[2012, 2012]' ] },
70        keysExamined: 480,
71        seeks: 1,
72        dupsTested: 0,
73        dupsDropped: 0
74      }
75    }
76  },
77  command: { find: 'movie', filter: { year: 2012 }, '$db': 'rottenMovies' },
78  serverInfo: {
79    host: 'Profile2022LARGE10',
80    port: 27017,
81    version: '6.0.3',
82    gitVersion: 'f803681c3ae19817d31958965850193de067c516'
83  },
84  serverParameters: {
85    internalQueryFacetBufferSizeBytes: 104857600,
86    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
87    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
88    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
89    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
90    internalQueryProhibitBlockingMergeOnMongoS: 0,

```



```

91     internalQueryMaxAddToSetBytes: 104857600,
92     internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
93 },
94 ok: 1,
95 '$clusterTime': {
96     clusterTime: Timestamp({ t: 1673285143, i: 1 }),
97     signature: {
98         hash: Binary(Buffer.from("00000000000000000000000000000000", "
99         hex"), 0),
100         keyId: Long("0")
101     }
102 },
103 operationTime: Timestamp({ t: 1673285143, i: 1 })

```

Listing 13: Test

## C.3 top critic rating

Before the index

```

1 {
2   explainVersion: '1',
3   queryPlanner: {
4     namespace: 'rottenMovies.movie',
5     indexFilterSet: false,
6     parsedQuery: {},
7     queryHash: '33018E32',
8     planCacheKey: '33018E32',
9     maxIndexedOrSolutionsReached: false,
10    maxIndexedAndSolutionsReached: false,
11    maxScansToExplodeReached: false,
12    winningPlan: {
13      stage: 'SORT',
14      sortPattern: { top_critic_rating: 1 },
15      memLimit: 104857600,
16      type: 'simple',
17      inputStage: { stage: 'COLLSCAN', direction: 'forward' }
18    },
19    rejectedPlans: []
20  },
21  executionStats: {
22    executionSuccess: true,
23    nReturned: 14104,
24    executionTimeMillis: 1818,
25    totalKeysExamined: 0,
26    totalDocsExamined: 14104,
27    executionStages: {
28      stage: 'SORT',
29      nReturned: 14104,
30      executionTimeMillisEstimate: 1740,
31      works: 28211,
32      advanced: 14104,
33      needTime: 14106,
34      needYield: 0,
35      saveState: 47,
36      restoreState: 47,
37      isEOF: 1,
38      sortPattern: { top_critic_rating: 1 },
39      memLimit: 104857600,

```

```

40     type: 'simple',
41     totalDataSizeSorted: 262640385,
42     usedDisk: true,
43     spills: 3,
44     inputStage: {
45         stage: 'COLLSCAN',
46         nReturned: 14104,
47         executionTimeMillisEstimate: 0,
48         works: 14106,
49         advanced: 14104,
50         needTime: 1,
51         needYield: 0,
52         saveState: 47,
53         restoreState: 47,
54         isEOF: 1,
55         direction: 'forward',
56         docsExamined: 14104
57     }
58 }
59 },
60 command: {
61     find: 'movie',
62     filter: {},
63     sort: { top_critic_rating: 1 },
64     '$db': 'rottenMovies'
65 },
66 serverInfo: {
67     host: 'Profile2022LARGE10',
68     port: 27017,
69     version: '6.0.3',
70     gitVersion: 'f803681c3ae19817d31958965850193de067c516'
71 },
72 serverParameters: {
73     internalQueryFacetBufferSizeBytes: 104857600,
74     internalQueryFacetMaxOutputDocSizeBytes: 104857600,
75     internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
76     internalDocumentSourceGroupMaxMemoryBytes: 104857600,
77     internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
78     internalQueryProhibitBlockingMergeOnMongoS: 0,
79     internalQueryMaxAddToSetBytes: 104857600,
80     internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
81 },
82 ok: 1,
83 '$clusterTime': {
84     clusterTime: Timestamp({ t: 1673287293, i: 1 }),
85     signature: {
86         hash: Binary(Buffer.from("000000000000000000000000000000000000", "
87         hex"), 0),
88         keyId: Long("0")
89     }
90 },
91 operationTime: Timestamp({ t: 1673287293, i: 1 })

```

Listing 14: Test

After the index

```

1 {
2     explainVersion: '1',
3     queryPlanner: {
4         namespace: 'rottenMovies.movie',

```

```

5     indexFilterSet: false ,
6     parsedQuery: {},
7     queryHash: '33018E32',
8     planCacheKey: '33018E32',
9     maxIndexedOrSolutionsReached: false ,
10    maxIndexedAndSolutionsReached: false ,
11    maxScansToExplodeReached: false ,
12    winningPlan: {
13        stage: 'FETCH',
14        inputStage: {
15            stage: 'IXSCAN',
16            keyPattern: { top_critic_rating: 1 },
17            indexName: 'top_critic_rating_1',
18            isMultiKey: false ,
19            multiKeyPaths: { top_critic_rating: [] },
20            isUnique: false ,
21            isSparse: false ,
22            isPartial: false ,
23            indexVersion: 2,
24            direction: 'forward',
25            indexBounds: { top_critic_rating: [ '[MinKey, MaxKey]' ] }
26        }
27    },
28    rejectedPlans: []
29 },
30 executionStats: {
31     executionSuccess: true ,
32     nReturned: 14104,
33     executionTimeMillis: 24,
34     totalKeysExamined: 14104,
35     totalDocsExamined: 14104,
36     executionStages: {
37         stage: 'FETCH',
38         nReturned: 14104,
39         executionTimeMillisEstimate: 5,
40         works: 14105,
41         advanced: 14104,
42         needTime: 0,
43         needYield: 0,
44         saveState: 14,
45         restoreState: 14,
46         isEOF: 1,
47         docsExamined: 14104,
48         alreadyHasObj: 0,
49         inputStage: {
50             stage: 'IXSCAN',
51             nReturned: 14104,
52             executionTimeMillisEstimate: 1,
53             works: 14105,
54             advanced: 14104,
55             needTime: 0,
56             needYield: 0,
57             saveState: 14,
58             restoreState: 14,
59             isEOF: 1,
60             keyPattern: { top_critic_rating: 1 },
61             indexName: 'top_critic_rating_1',
62             isMultiKey: false ,
63             multiKeyPaths: { top_critic_rating: [] },
64             isUnique: false ,
65             isSparse: false ,

```

```

66         isPartial: false,
67         indexVersion: 2,
68         direction: 'forward',
69         indexBounds: { top_critic_rating: [ '[MinKey, MaxKey]' ] },
70         keysExamined: 14104,
71         seeks: 1,
72         dupsTested: 0,
73         dupsDropped: 0
74     }
75 }
76 },
77 command: {
78     find: 'movie',
79     filter: {},
80     sort: { top_critic_rating: 1 },
81     '$db': 'rottenMovies'
82 },
83 serverInfo: {
84     host: 'Profile2022LARGE10',
85     port: 27017,
86     version: '6.0.3',
87     gitVersion: 'f803681c3ae19817d31958965850193de067c516'
88 },
89 serverParameters: {
90     internalQueryFacetBufferSizeBytes: 104857600,
91     internalQueryFacetMaxOutputDocSizeBytes: 104857600,
92     internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
93     internalDocumentSourceGroupMaxMemoryBytes: 104857600,
94     internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
95     internalQueryProhibitBlockingMergeOnMongoS: 0,
96     internalQueryMaxAddToSetBytes: 104857600,
97     internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
98 },
99 ok: 1,
100 '$clusterTime': {
101     clusterTime: Timestamp({ t: 1673285423, i: 1 }),
102     signature: {
103         hash: Binary(Buffer.from("000000000000000000000000000000000000",
104             hex"), 0),
105         keyId: Long("0")
106     }
107 },
108 operationTime: Timestamp({ t: 1673285423, i: 1 })

```

Listing 15: Test

## C.4 user rating

Before the index

```

1 {
2     explainVersion: '1',
3     queryPlanner: {
4         namespace: 'rottenMovies.movie',
5         indexFilterSet: false,
6         parsedQuery: {},
7         queryHash: '3E9B1E6C',
8         planCacheKey: '3E9B1E6C',
9         maxIndexedOrSolutionsReached: false,

```

```

10     maxIndexedAndSolutionsReached: false ,
11     maxScansToExplodeReached: false ,
12     winningPlan: {
13         stage: 'SORT',
14         sortPattern: { user_rating: 1 },
15         memLimit: 104857600,
16         type: 'simple',
17         inputStage: { stage: 'COLLSCAN', direction: 'forward' }
18     },
19     rejectedPlans: []
20 },
21 executionStats: {
22     executionSuccess: true ,
23     nReturned: 14104,
24     executionTimeMillis: 1779,
25     totalKeysExamined: 0,
26     totalDocsExamined: 14104,
27     executionStages: {
28         stage: 'SORT',
29         nReturned: 14104,
30         executionTimeMillisEstimate: 1698,
31         works: 28211,
32         advanced: 14104,
33         needTime: 14106,
34         needYield: 0,
35         saveState: 45,
36         restoreState: 45,
37         isEOF: 1,
38         sortPattern: { user_rating: 1 },
39         memLimit: 104857600,
40         type: 'simple',
41         totalDataSizeSorted: 262640385,
42         usedDisk: true ,
43         spills: 3,
44         inputStage: {
45             stage: 'COLLSCAN',
46             nReturned: 14104,
47             executionTimeMillisEstimate: 0,
48             works: 14106,
49             advanced: 14104,
50             needTime: 1,
51             needYield: 0,
52             saveState: 45,
53             restoreState: 45,
54             isEOF: 1,
55             direction: 'forward',
56             docsExamined: 14104
57         }
58     }
59 },
60 command: {
61     find: 'movie',
62     filter: {},
63     sort: { user_rating: 1 },
64     '$db': 'rottenMovies'
65 },
66 serverInfo: {
67     host: 'Profile2022LARGE10',
68     port: 27017,
69     version: '6.0.3',
70     gitVersion: 'f803681c3ae19817d31958965850193de067c516'

```

```

71 },
72 serverParameters: {
73     internalQueryFacetBufferSizeBytes: 104857600,
74     internalQueryFacetMaxOutputDocSizeBytes: 104857600,
75     internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
76     internalDocumentSourceGroupMaxMemoryBytes: 104857600,
77     internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
78     internalQueryProhibitBlockingMergeOnMongoS: 0,
79     internalQueryMaxAddToSetBytes: 104857600,
80     internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
81 },
82 ok: 1,
83 '$clusterTime': {
84     clusterTime: Timestamp({ t: 1673287353, i: 1 }),
85     signature: {
86         hash: Binary(Buffer.from("000000000000000000000000000000000000",
87             hex"), 0),
88         keyId: Long("0")
89     }
90 },
91 operationTime: Timestamp({ t: 1673287353, i: 1 })
92 }

```

Listing 16: Test

After the index

```

1 {
2     explainVersion: '1',
3     queryPlanner: {
4         namespace: 'rottenMovies.movie',
5         indexFilterSet: false,
6         parsedQuery: {},
7         queryHash: '3E9B1E6C',
8         planCacheKey: '3E9B1E6C',
9         maxIndexedOrSolutionsReached: false,
10        maxIndexedAndSolutionsReached: false,
11        maxScansToExplodeReached: false,
12        winningPlan: {
13            stage: 'FETCH',
14            inputStage: {
15                stage: 'IXSCAN',
16                keyPattern: { user_rating: 1 },
17                indexName: 'user_rating_1',
18                isMultiKey: false,
19                multiKeyPaths: { user_rating: [] },
20                isUnique: false,
21                isSparse: false,
22                isPartial: false,
23                indexVersion: 2,
24                direction: 'forward',
25                indexBounds: { user_rating: [ '[MinKey, MaxKey]' ] }
26            }
27        },
28        rejectedPlans: []
29    },
30    executionStats: {
31        executionSuccess: true,
32        nReturned: 14104,
33        executionTimeMillis: 25,
34        totalKeysExamined: 14104,
35        totalDocsExamined: 14104,

```

```

36     executionStages: {
37         stage: 'FETCH',
38         nReturned: 14104,
39         executionTimeMillisEstimate: 5,
40         works: 14105,
41         advanced: 14104,
42         needTime: 0,
43         needYield: 0,
44         saveState: 14,
45         restoreState: 14,
46         isEOF: 1,
47         docsExamined: 14104,
48         alreadyHasObj: 0,
49         inputStage: {
50             stage: 'IXSCAN',
51             nReturned: 14104,
52             executionTimeMillisEstimate: 2,
53             works: 14105,
54             advanced: 14104,
55             needTime: 0,
56             needYield: 0,
57             saveState: 14,
58             restoreState: 14,
59             isEOF: 1,
60             keyPattern: { user_rating: 1 },
61             indexName: 'user_rating_1',
62             isMultiKey: false,
63             multiKeyPaths: { user_rating: [ ] },
64             isUnique: false,
65             isSparse: false,
66             isPartial: false,
67             indexVersion: 2,
68             direction: 'forward',
69             indexBounds: { user_rating: [ '[MinKey, MaxKey]' ] },
70             keysExamined: 14104,
71             seeks: 1,
72             dupsTested: 0,
73             dupsDropped: 0
74         }
75     },
76 },
77 command: {
78     find: 'movie',
79     filter: {},
80     sort: { user_rating: 1 },
81     '$db': 'rottenMovies'
82 },
83 serverInfo: {
84     host: 'Profile2022LARGE10',
85     port: 27017,
86     version: '6.0.3',
87     gitVersion: 'f803681c3ae19817d31958965850193de067c516'
88 },
89 serverParameters: {
90     internalQueryFacetBufferSizeBytes: 104857600,
91     internalQueryFacetMaxOutputDocSizeBytes: 104857600,
92     internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
93     internalDocumentSourceGroupMaxMemoryBytes: 104857600,
94     internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
95     internalQueryProhibitBlockingMergeOnMongoS: 0,
96     internalQueryMaxAddToSetBytes: 104857600,

```

```

97     internalDocumentSourceSet WindowFieldsMaxMemoryBytes: 104857600
98   },
99   ok: 1,
100   '$clusterTime': {
101     clusterTime: Timestamp({ t: 1673285383, i: 1 }),
102     signature: {
103       hash: Binary(Buffer.from("00000000000000000000000000000000", "
104       hex"), 0),
105       keyId: Long("0")
106     }
107   },
108   operationTime: Timestamp({ t: 1673285383, i: 1 })

```

Listing 17: Test

## C .5 personnel.primaryName

Before the index

```

1  {
2    explainVersion: '1',
3    queryPlanner: {
4      namespace: 'rottenMovies.movie',
5      indexFilterSet: false,
6      parsedQuery: { 'personnel.primaryName': { '$eq': '' } },
7      queryHash: 'E212F03B',
8      planCacheKey: 'E212F03B',
9      maxIndexedOrSolutionsReached: false,
10     maxIndexedAndSolutionsReached: false,
11     maxScansToExplodeReached: false,
12     winningPlan: {
13       stage: 'COLLSCAN',
14       filter: { 'personnel.primaryName': { '$eq': '' } },
15       direction: 'forward'
16     },
17     rejectedPlans: []
18   },
19   executionStats: {
20     executionSuccess: true,
21     nReturned: 0,
22     executionTimeMillis: 47,
23     totalKeysExamined: 0,
24     totalDocsExamined: 14104,
25     executionStages: {
26       stage: 'COLLSCAN',
27       filter: { 'personnel.primaryName': { '$eq': '' } },
28       nReturned: 0,
29       executionTimeMillisEstimate: 9,
30       works: 14106,
31       advanced: 0,
32       needTime: 14105,
33       needYield: 0,
34       saveState: 14,
35       restoreState: 14,
36       isEOF: 1,
37       direction: 'forward',
38       docsExamined: 14104
39     }
40   },

```



```

41  command: {
42    find: 'movie',
43    filter: { 'personnel.primaryName': '' },
44    '$db': 'rottenMovies'
45  },
46  serverInfo: {
47    host: 'Profile2022LARGE10',
48    port: 27017,
49    version: '6.0.3',
50    gitVersion: 'f803681c3ae19817d31958965850193de067c516'
51  },
52  serverParameters: {
53    internalQueryFacetBufferSizeBytes: 104857600,
54    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
55    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
56    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
57    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
58    internalQueryProhibitBlockingMergeOnMongoS: 0,
59    internalQueryMaxAddToSetBytes: 104857600,
60    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
61  },
62  ok: 1,
63  '$clusterTime': {
64    clusterTime: Timestamp({ t: 1673287593, i: 1 }),
65    signature: {
66      hash: Binary(Buffer.from("0000000000000000000000000000000000", "
67      hex"), 0),
68      keyId: Long("0")
69    }
70  },
71  operationTime: Timestamp({ t: 1673287593, i: 1 })

```

Listing 18: Test

After the index

```

1  {
2    explainVersion: '1',
3    queryPlanner: {
4      namespace: 'rottenMovies.movie',
5      indexFilterSet: false,
6      parsedQuery: { 'personnel.primaryName': { '$eq': '' } },
7      queryHash: 'E212F03B',
8      planCacheKey: '9D4A6814',
9      maxIndexedOrSolutionsReached: false,
10     maxIndexedAndSolutionsReached: false,
11     maxScansToExplodeReached: false,
12     winningPlan: {
13       stage: 'FETCH',
14       inputStage: {
15         stage: 'IXSCAN',
16         keyPattern: { 'personnel.primaryName': 1 },
17         indexName: 'personnel.primaryName_1',
18         isMultiKey: true,
19         multiKeyPaths: { 'personnel.primaryName': [ 'personnel' ] },
20         isUnique: false,
21         isSparse: false,
22         isPartial: false,
23         indexVersion: 2,
24         direction: 'forward',
25         indexBounds: { 'personnel.primaryName': [ '[', '' ] }

```

```

26     }
27 },
28     rejectedPlans: []
29 },
30     executionStats: {
31         executionSuccess: true,
32         nReturned: 0,
33         executionTimeMillis: 0,
34         totalKeysExamined: 0,
35         totalDocsExamined: 0,
36         executionStages: {
37             stage: 'FETCH',
38             nReturned: 0,
39             executionTimeMillisEstimate: 0,
40             works: 1,
41             advanced: 0,
42             needTime: 0,
43             needYield: 0,
44             saveState: 0,
45             restoreState: 0,
46             isEOF: 1,
47             docsExamined: 0,
48             alreadyHasObj: 0,
49             inputStage: {
50                 stage: 'IXSCAN',
51                 nReturned: 0,
52                 executionTimeMillisEstimate: 0,
53                 works: 1,
54                 advanced: 0,
55                 needTime: 0,
56                 needYield: 0,
57                 saveState: 0,
58                 restoreState: 0,
59                 isEOF: 1,
60                 keyPattern: { 'personnel.primaryName': 1 },
61                 indexName: 'personnel.primaryName_1',
62                 isMultiKey: true,
63                 multiKeyPaths: { 'personnel.primaryName': [ 'personnel' ] },
64                 isUnique: false,
65                 isSparse: false,
66                 isPartial: false,
67                 indexVersion: 2,
68                 direction: 'forward',
69                 indexBounds: { 'personnel.primaryName': [ '["", ""]' ] },
70                 keysExamined: 0,
71                 seeks: 1,
72                 dupsTested: 0,
73                 dupsDropped: 0
74             }
75         }
76     },
77     command: {
78         find: 'movie',
79         filter: { 'personnel.primaryName': '' },
80         '$db': 'rottenMovies'
81     },
82     serverInfo: {
83         host: 'Profile2022LARGE10',
84         port: 27017,
85         version: '6.0.3',
86         gitVersion: 'f803681c3ae19817d31958965850193de067c516'

```

```

87  },
88  serverParameters: {
89    internalQueryFacetBufferSizeBytes: 104857600,
90    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
91    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
92    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
93    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
94    internalQueryProhibitBlockingMergeOnMongoS: 0,
95    internalQueryMaxAddToSetBytes: 104857600,
96    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
97  },
98  ok: 1,
99  '$clusterTime': {
100    clusterTime: Timestamp({ t: 1673287763, i: 1 }),
101    signature: {
102      hash: Binary(Buffer.from("0000000000000000000000000000000000", "
103      hex"), 0),
104      keyId: Long("0")
105    }
106  },
107  operationTime: Timestamp({ t: 1673287763, i: 1 })

```

Listing 19: Test

## D MongoDB indexes:User collection

### D.1 username

Before the index

```

1  {
2    explainVersion: '1',
3    queryPlanner: {
4      namespace: 'rottenMovies.user',
5      indexFilterSet: false,
6      parsedQuery: { username: { '$eq': 'Abbie Bernstein' } } },
7      queryHash: '7D9BB680',
8      planCacheKey: '7D9BB680',
9      maxIndexedOrSolutionsReached: false,
10     maxIndexedAndSolutionsReached: false,
11     maxScansToExplodeReached: false,
12     winningPlan: {
13       stage: 'COLLSCAN',
14       filter: { username: { '$eq': 'Abbie Bernstein' } } },
15       direction: 'forward'
16     },
17     rejectedPlans: []
18   },
19   executionStats: {
20     executionSuccess: true,
21     nReturned: 1,
22     executionTimeMillis: 6,
23     totalKeysExamined: 0,
24     totalDocsExamined: 8339,
25     executionStages: {
26       stage: 'COLLSCAN',
27       filter: { username: { '$eq': 'Abbie Bernstein' } } },
28       nReturned: 1,

```

```

29     executionTimeMillisEstimate: 0,
30     works: 8341,
31     advanced: 1,
32     needTime: 8339,
33     needYield: 0,
34     saveState: 8,
35     restoreState: 8,
36     isEOF: 1,
37     direction: 'forward',
38     docsExamined: 8339
39   }
40 },
41 command: {
42   find: 'user',
43   filter: { username: 'Abbie Bernstein' },
44   '$db': 'rottenMovies'
45 },
46 serverInfo: {
47   host: 'Profile2022LARGE10',
48   port: 27017,
49   version: '6.0.3',
50   gitVersion: 'f803681c3ae19817d31958965850193de067c516'
51 },
52 serverParameters: {
53   internalQueryFacetBufferSizeBytes: 104857600,
54   internalQueryFacetMaxOutputDocSizeBytes: 104857600,
55   internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
56   internalDocumentSourceGroupMaxMemoryBytes: 104857600,
57   internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
58   internalQueryProhibitBlockingMergeOnMongoS: 0,
59   internalQueryMaxAddToSetBytes: 104857600,
60   internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
61 },
62 ok: 1,
63 '$clusterTime': {
64   clusterTime: Timestamp({ t: 1673280753, i: 1 }),
65   signature: {
66     hash: Binary(Buffer.from("000000000000000000000000000000000000",
67     hex"), 0),
68     keyId: Long("0")
69   }
70 },
71 operationTime: Timestamp({ t: 1673280753, i: 1 })

```

Listing 20: Test

## After the index

```

1 {
2   explainVersion: '1',
3   queryPlanner: {
4     namespace: 'rottenMovies.user',
5     indexFilterSet: false,
6     parsedQuery: { username: { '$eq': 'Abbie Bernstein' } },
7     queryHash: '7D9BB680',
8     planCacheKey: '24069050',
9     maxIndexedOrSolutionsReached: false,
10    maxIndexedAndSolutionsReached: false,
11    maxScansToExplodeReached: false,
12    winningPlan: {
13      stage: 'FETCH',

```

```

14     inputStage: {
15         stage: 'IXSCAN',
16         keyPattern: { username: 1 },
17         indexName: 'username_1',
18         isMultiKey: false,
19         multiKeyPaths: { username: [] },
20         isUnique: false,
21         isSparse: false,
22         isPartial: false,
23         indexVersion: 2,
24         direction: 'forward',
25         indexBounds: { username: [ '["Abbie Bernstein", "Abbie Bernstein"]' ]
26     }
27 },
28 rejectedPlans: []
29 },
30 executionStats: {
31     executionSuccess: true,
32     nReturned: 1,
33     executionTimeMillis: 1,
34     totalKeysExamined: 1,
35     totalDocsExamined: 1,
36     executionStages: {
37         stage: 'FETCH',
38         nReturned: 1,
39         executionTimeMillisEstimate: 1,
40         works: 2,
41         advanced: 1,
42         needTime: 0,
43         needYield: 0,
44         saveState: 0,
45         restoreState: 0,
46         isEOF: 1,
47         docsExamined: 1,
48         alreadyHasObj: 0,
49         inputStage: {
50             stage: 'IXSCAN',
51             nReturned: 1,
52             executionTimeMillisEstimate: 1,
53             works: 2,
54             advanced: 1,
55             needTime: 0,
56             needYield: 0,
57             saveState: 0,
58             restoreState: 0,
59             isEOF: 1,
60             keyPattern: { username: 1 },
61             indexName: 'username_1',
62             isMultiKey: false,
63             multiKeyPaths: { username: [] },
64             isUnique: false,
65             isSparse: false,
66             isPartial: false,
67             indexVersion: 2,
68             direction: 'forward',
69             indexBounds: { username: [ '["Abbie Bernstein", "Abbie Bernstein"]' ]
70         },
71         keysExamined: 1,
72         seeks: 1,
73         dupsTested: 0,

```

```

73     dupsDropped: 0
74   }
75 }
76 },
77 command: {
78   find: 'user',
79   filter: { username: 'Abbie Bernstein' },
80   '$db': 'rottenMovies'
81 },
82 serverInfo: {
83   host: 'Profile2022LARGE10',
84   port: 27017,
85   version: '6.0.3',
86   gitVersion: 'f803681c3ae19817d31958965850193de067c516'
87 },
88 serverParameters: {
89   internalQueryFacetBufferSizeBytes: 104857600,
90   internalQueryFacetMaxOutputDocSizeBytes: 104857600,
91   internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
92   internalDocumentSourceGroupMaxMemoryBytes: 104857600,
93   internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
94   internalQueryProhibitBlockingMergeOnMongoS: 0,
95   internalQueryMaxAddToSetBytes: 104857600,
96   internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
97 },
98 ok: 1,
99 '$clusterTime': {
100   clusterTime: Timestamp({ t: 1673285013, i: 1 }),
101   signature: {
102     hash: Binary(Buffer.from("000000000000000000000000000000000000", "
103     hex"), 0),
104     keyId: Long("0")
105   }
106 },
107 operationTime: Timestamp({ t: 1673285013, i: 1 })

```

Listing 21: Test

## D .2 date of birth

Before the index

```

1 {
2   explainVersion: '1',
3   queryPlanner: {
4     namespace: 'rottenMovies.user',
5     indexFilterSet: false,
6     parsedQuery: {
7       date_of_birth: {
8         '$eq': 'Mon Jan 09 2023 17:05:07 GMT+0000 (Western European Standard
9         Time)'
10      }
11    },
12    queryHash: 'D7A0117C',
13    planCacheKey: 'D7A0117C',
14    maxIndexedOrSolutionsReached: false,
15    maxIndexedAndSolutionsReached: false,
16    maxScansToExplodeReached: false,
17    winningPlan: {

```

```

17     stage: 'COLLSCAN',
18     filter: {
19         date_of_birth: {
20             '$eq': 'Mon Jan 09 2023 17:05:07 GMT+0000 (Western European
Standard Time)'
21         }
22     },
23     direction: 'forward'
24 },
25     rejectedPlans: []
26 },
27     executionStats: {
28         executionSuccess: true,
29         nReturned: 0,
30         executionTimeMillis: 32,
31         totalKeysExamined: 0,
32         totalDocsExamined: 8339,
33         executionStages: {
34             stage: 'COLLSCAN',
35             filter: {
36                 date_of_birth: {
37                     '$eq': 'Mon Jan 09 2023 17:05:07 GMT+0000 (Western European
Standard Time)'
38                 }
39             },
40             nReturned: 0,
41             executionTimeMillisEstimate: 23,
42             works: 8341,
43             advanced: 0,
44             needTime: 8340,
45             needYield: 0,
46             saveState: 8,
47             restoreState: 8,
48             isEOF: 1,
49             direction: 'forward',
50             docsExamined: 8339
51         }
52     },
53     command: {
54         find: 'user',
55         filter: {
56             date_of_birth: 'Mon Jan 09 2023 17:05:07 GMT+0000 (Western European
Standard Time)'
57         },
58         '$db': 'rottenMovies'
59     },
60     serverInfo: {
61         host: 'Profile2022LARGE10',
62         port: 27017,
63         version: '6.0.3',
64         gitVersion: 'f803681c3ae19817d31958965850193de067c516'
65     },
66     serverParameters: {
67         internalQueryFacetBufferSizeBytes: 104857600,
68         internalQueryFacetMaxOutputDocSizeBytes: 104857600,
69         internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
70         internalDocumentSourceGroupMaxMemoryBytes: 104857600,
71         internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
72         internalQueryProhibitBlockingMergeOnMongoS: 0,
73         internalQueryMaxAddToSetBytes: 104857600,
74         internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600

```

```

75 },
76 ok: 1,
77 '$clusterTime': {
78   clusterTime: Timestamp({ t: 1673283903, i: 1 }),
79   signature: {
80     hash: Binary(Buffer.from("00000000000000000000000000000000", "
81     hex"), 0),
82     keyId: Long("0")
83   }
84 },
85 operationTime: Timestamp({ t: 1673283903, i: 1 })
86 }

```

Listing 22: Test

After the index

```

1 {
2   explainVersion: '1',
3   queryPlanner: {
4     namespace: 'rottenMovies.user',
5     indexFilterSet: false,
6     parsedQuery: {
7       date_of_birth: {
8         '$eq': 'Mon Jan 09 2023 17:24:42 GMT+0000 (Western European Standard
9         Time)'
10      }
11    },
12    queryHash: 'D7A0117C',
13    planCacheKey: '90F68BB6',
14    maxIndexedOrSolutionsReached: false,
15    maxIndexedAndSolutionsReached: false,
16    maxScansToExplodeReached: false,
17    winningPlan: {
18      stage: 'FETCH',
19      inputStage: {
20        stage: 'IXSCAN',
21        keyPattern: { date_of_birth: 1 },
22        indexName: 'date_of_birth_1',
23        isMultiKey: false,
24        multiKeyPaths: { date_of_birth: [] },
25        isUnique: false,
26        isSparse: false,
27        isPartial: false,
28        indexVersion: 2,
29        direction: 'forward',
30        indexBounds: {
31          date_of_birth: [
32            '["Mon Jan 09 2023 17:24:42 GMT+0000 (Western European Standard
33            Time)", "Mon Jan 09 2023 17:24:42 GMT+0000 (Western European Standard
34            Time)"]',
35          ]
36        }
37      },
38      rejectedPlans: []
39    },
40    executionStats: {
41      executionSuccess: true,
42      nReturned: 0,
43      executionTimeMillis: 1,
44      totalKeysExamined: 0,

```



```

43     totalDocsExamined: 0,
44     executionStages: {
45         stage: 'FETCH',
46         nReturned: 0,
47         executionTimeMillisEstimate: 0,
48         works: 1,
49         advanced: 0,
50         needTime: 0,
51         needYield: 0,
52         saveState: 0,
53         restoreState: 0,
54         isEOF: 1,
55         docsExamined: 0,
56         alreadyHasObj: 0,
57         inputStage: {
58             stage: 'IXSCAN',
59             nReturned: 0,
60             executionTimeMillisEstimate: 0,
61             works: 1,
62             advanced: 0,
63             needTime: 0,
64             needYield: 0,
65             saveState: 0,
66             restoreState: 0,
67             isEOF: 1,
68             keyPattern: { date_of_birth: 1 },
69             indexName: 'date_of_birth_1',
70             isMultiKey: false,
71             multiKeyPaths: { date_of_birth: [] },
72             isUnique: false,
73             isSparse: false,
74             isPartial: false,
75             indexVersion: 2,
76             direction: 'forward',
77             indexBounds: {
78                 date_of_birth: [
79                     '["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)"]',
80                 ]
81             },
82             keysExamined: 0,
83             seeks: 1,
84             dupsTested: 0,
85             dupsDropped: 0
86         }
87     }
88 },
89 command: {
90     find: 'user',
91     filter: {
92         date_of_birth: 'Mon Jan 09 2023 17:24:42 GMT+0000 (Western European
Standard Time)'
93     },
94     '$db': 'rottenMovies'
95 },
96 serverInfo: {
97     host: 'Profile2022LARGE10',
98     port: 27017,
99     version: '6.0.3',
100    gitVersion: 'f803681c3ae19817d31958965850193de067c516'

```

```

101 },
102 serverParameters: {
103     internalQueryFacetBufferSizeBytes: 104857600,
104     internalQueryFacetMaxOutputDocSizeBytes: 104857600,
105     internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
106     internalDocumentSourceGroupMaxMemoryBytes: 104857600,
107     internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
108     internalQueryProhibitBlockingMergeOnMongoS: 0,
109     internalQueryMaxAddToSetBytes: 104857600,
110     internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
111 },
112 ok: 1,
113 '$clusterTime': {
114     clusterTime: Timestamp({ t: 1673285073, i: 1 }),
115     signature: {
116         hash: Binary(Buffer.from("00000000000000000000000000000000", "
117         hex"), 0),
118         keyId: Long("0")
119     }
120 },
121 operationTime: Timestamp({ t: 1673285073, i: 1 })
122 }

```

Listing 23: Test

## E Application code

### E.1 Pom.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.
   org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache
   .org/xsd/maven-4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <parent>
6         <groupId>org.springframework.boot</groupId>
7         <artifactId>spring-boot-starter-parent</artifactId>
8         <version>3.0.0</version>
9         <relativePath/> <!-- lookup parent from repository -->
10    </parent>
11    <groupId>it.unipi.dii.lsmsdb</groupId>
12    <artifactId>rottenMovies</artifactId>
13    <version>0.0.1-SNAPSHOT</version>
14    <name>rottenMovies</name>
15    <description>Project for the rotten movies service</description>
16    <properties>
17        <java.version>19</java.version>
18    </properties>
19    <dependencies>
20        <dependency>
21            <groupId>org.springframework.boot</groupId>
22            <artifactId>spring-boot-starter-thymeleaf</artifactId>
23        </dependency>
24        <dependency>
25            <groupId>org.springframework.boot</groupId>
26            <artifactId>spring-boot-starter-web</artifactId>
27        </dependency>

```

```

28     <dependency>
29         <groupId>org.springframework.boot</groupId>
30         <artifactId>spring-boot-starter-data-mongodb</artifactId>
31     </dependency>
32     <dependency>
33         <groupId>org.springframework.boot</groupId>
34         <artifactId>spring-boot-starter-data-mongodb-reactive</artifactId>
35     </dependency>
36     <dependency>
37         <groupId>org.springframework.boot</groupId>
38         <artifactId>spring-boot-starter-data-neo4j</artifactId>
39     </dependency>
40
41     <dependency>
42         <groupId>org.springframework.boot</groupId>
43         <artifactId>spring-boot-starter-test</artifactId>
44         <scope>test</scope>
45     </dependency>
46     <dependency>
47         <groupId>io.projectreactor</groupId>
48         <artifactId>reactor-test</artifactId>
49         <scope>test</scope>
50     </dependency>
51     <dependency>
52         <groupId>com.google.code.gson</groupId>
53         <artifactId>gson</artifactId>
54         <version>2.10</version>
55     </dependency>
56     <dependency>
57         <groupId>com.fasterxml.jackson.core</groupId>
58         <artifactId>jackson-annotations</artifactId>
59         <version>2.14.1</version>
60     </dependency>
61     <dependency>
62         <groupId>com.fasterxml.jackson.core</groupId>
63         <artifactId>jackson-databind</artifactId>
64         <version>2.14.0</version>
65     </dependency>
66     <dependency>
67         <groupId>org.neo4j.driver</groupId>
68         <artifactId>neo4j-java-driver</artifactId>
69         <version>5.3.0</version>
70     </dependency>
71 </dependencies>
72
73 <build>
74     <plugins>
75         <plugin>
76             <groupId>org.springframework.boot</groupId>
77             <artifactId>spring-boot-maven-plugin</artifactId>
78         </plugin>
79     </plugins>
80 </build>
81
82 </project>

```

Listing 24: Test