A Python Code

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

```
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', nrows=numberofrows, header=0)
7 title_principals = pd.read_csv("/content/drive/MyDrive/Dataset/Original/
      title principals.tsv", nrows=numberofrows, sep='\t', header=0)
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
print (title basics.head (3))
15 merged1=pd.merge(title_basics, title_principals, how='inner', on='tconst')
16 del title_basics, title_principals
17 print (merged1)
19 name basics=pd.read csv("/content/drive/MyDrive/Dataset/Original/name basics.
      tsv", sep='\t', nrows=numberofrows, header=0)
21 merged 2=pd . merge (merged 1, name basics, how='inner', on='nconst')
22 del merged1
23 merged2=merged2.drop(columns=["ordering","nconst","birthYear","deathYear","
      knownForTitles", "primaryProfession"])
24 del name basics
print (merged 2)
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')
result=merged2.drop_duplicates(subset=['tconst'])
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)
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 number of rows=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)
s to keep = ["rotten tomatoes link", "movie title", "production company","
      critics consensus",
              "tomatometer status", "tomatometer rating", "tomatometer count",
              "audience status", "audience rating", "audience count",
10
              "tomatometer top critics count", "tomatometer fresh critics count"
11
              "tomatometer rotten critics count"]
12
13
14 movies = movies [to keep]
15 to drop = ["publisher name"]
16 reviews=reviews.drop(columns=to_drop)
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:
    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:
    result = result \cdot drop([x], axis = 1)
29 for x in arr:
    result=result.merge(categories[x],on='rotten tomatoes link')
30
31
32 print (result)
_{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

```
merged[x]=pd.merge(imdb,rotten,how='inner', left on=x, right on=
     movie title')
    print(x)
14
    merged[x] = merged[x].drop duplicates(subset = [x])
15
    merged[x]=merged[x].drop(columns=['titleType', 'tomatometer count', '
     tomatometer_top_critics_count'])
    merged[x]=merged[x].rename(columns={'startYear': 'year'})
17
    merged[x]=merged[x].drop(columns=['rotten_tomatoes_link', 'movie_title']+[j
18
      for j in choose if j!=x])
    print(len(pd.unique(merged[x][x])))
19
    print(list(merged[x]))
20
    print ("=
21
    merged[x].to csv(f"/content/drive/MyDrive/Dataset/ImdbJoinRotten{x}.csv",
     index=False)
    merged[x] = merged[x]. head (rowHeadDataset)
23
    merged[x].to csv(f"/content/drive/MyDrive/Dataset/headDataset{x}.csv",index
24
     =False)
```

Listing 3: Test

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

```
2 import pandas as pd
3 from ast import literal eval
4 from google.colab import drive
5 drive.mount('/content/drive')
7 number of rows=None
s df = pd.read csv("/content/drive/MyDrive/Dataset/ImdbJoinRottenprimaryTitle.
     csv", nrows=numberofrows, header=0)
10 #print ([x.split(',') for x in df['genres']])
11 print (df)
13 col = ["primaryName", "category", "job", "characters"]
14 col1 = ["critic name", "top critic", "review type", "review score", "review date"
      , "review content"]
15
df['personnel'] = ""
17 df['review'] = ""
18
19 for row in range (df [col [0]]. size):
    it = df['genres'][row]
    df['genres'][row] = ['"' + x + '"' for x in it.split(',')] if it != '\\N'
21
     else
    tmp = []
22
    for c in col:
23
      tmp.append({c:eval(df[c][row])})
24
    for c in range(len(tmp[0][col[0]])):
26
      res. append (\{\})
27
    for i, j in zip(col, tmp):
28
      for idx, x in enumerate(j[i]):
29
        30
31
          if i == 'characters':
32
33
            x = eval(x)
```

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

```
dbname = get database()
       collection = dbname['user']
12
       total = collection.count documents({})
13
       for i, user in enumerate(collection.find()):
14
           all reviews = user['last_3_reviews']
15
           sorted list = sorted(all reviews, key=lambda t: t['review date'])
16
      [-3:]
17
           hashed = hashlib.md5(user["username"].encode()).hexdigest()
18
19
           newvalues = { "$set": { 'password': hashed, 'last 3 reviews':
20
      sorted list } }
           filter = { 'username': user['username']}
21
           collection.update_one(filter, newvalues)
print(f"{i/total:%}\r", end='')
22
23
       print()
24
```

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
5 def get_database():
     CONNECTION STRING = "mongodb: //localhost: 27017"
     client = Mongo Client (CONNECTION STRING)
     return client['rottenMovies']
10 class Neo4jGraph:
11
      def __init__(self , uri , user , password):
12
           self.driver = GraphDatabase.driver(uri, auth=(user, password),
13
      database="rottenmoviesgraphdb")
14
      def close (self):
15
16
           self.driver.close()
17
      def addUser(self , uid , name , isTop):
18
           with self.driver.session() as session:
19
               if isTop:
20
                   result = session.execute write(self. addTopCritic, uid, name)
21
               else
22
                   result = session.execute write(self. addUser, uid, name)
24
      def addMovie(self, mid, title):
25
           with self.driver.session() as session:
26
               result = session.execute write(self. addMovie, mid, title)
27
28
      def addReview(self, name, mid, freshness, content, date):
29
           with self.driver.session() as session:
30
               result = session.execute_write(self._addReview, name, mid,
31
      freshness, content, date)
32
      def addFollow(self, uid, cid):
33
           with self.driver.session() as session:
34
               result = session.execute write(self. addFollow, uid, cid)
35
36
```

```
@staticmethod
      def addUser(tx, uid, name):
38
           query = "CREATE (n: User{id: \"" + str(uid) + "\", name: \"" + name.
39
      replace('"', '\\"') + "\"})"
           #print (query)
40
           result = tx.run(query)
41
42
       @staticmethod
43
       def addTopCritic(tx, cid, name):
44
           query = "CREATE(m: Top Critic \{id: \"" + str(cid) + "\", name: \"" + name.
45
      replace('"', '\\"') + "\"})"
           #print (query)
46
           result = tx.run(query)
47
48
       @staticmethod
49
      def addMovie(tx, mid, title):
50
           query = "CREATE(o:Movie{id:\"" + str(mid) + "\", title:\"" + title.
51
      replace('"', '\\"') + "\"})"
           #print (query)
52
           result = tx.run(query)
53
54
       @staticmethod
55
       def addReview(tx, name, mid, freshness, content, date): # date in format
56
       YYYY-mm-dd, freshness in [TRUE, FALSE]
           query = "MATCH(n\{name: \"" + str(name).replace('"', '\\"') + "\"'\}), (m)
57
      :Movie\{id: \ "" + str(mid) + "\"\}\} CREATE (n)-[r:REVIEWED\{freshness: "+
      freshness + ", date:date('" + date + "'), content:\"" + content.replace('
      "', '\\"') + "\"\}]->(m)"
           #print (query)
58
           result = tx.run(query)
59
60
       @staticmethod
61
      def addFollow(tx, uid, cid):
62
           query = "MATCH(n: User\{id: \"" + str(uid) + "\"\}), (m: Top Critic\{id: \""
63
      + \operatorname{str}(\operatorname{cid}) + "\"\} CREATE (n) - [\operatorname{r}: FOLLOWS] - > (m) "
64
           #print (query)
           result = tx.run(query)
65
66
     _{\rm name} = "_{\rm main} :
67 if
      # dbs initialization
68
      dbname = get database()
69
      graphDB = Neo4jGraph("bolt://localhost:7687", "neo4j", "password")
70
71
      # user creation
72
       collection = dbname['user']
73
       total = collection.count documents({})
74
       print(f"user {total = }")
75
       for i, user in enumerate(list(collection.find({}), {" id":1, "username":1,
76
       "date of birth":1}))):
           graphDB.addUser(user['id'], user['username'], 'date of birth' not in
       user)
           if not i %100:
78
                print (f'' \{ (i+1)/t \text{ ot al} : \% \} \setminus r'', \text{ end} = ',')
79
80
      # movie creation and review linking
81
       collection = dbname['movie']
82
       total = collection.count documents({})
83
       print(f"\nmovie {total = }")
84
      for i, movie in enumerate(list(collection.find({}, {" id":1, "
85
      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']:
88
                        graphDB.addReview(rev['critic_name'], movie['_id'], {"Fresh":"
89
         TRUE", "Rotten": "FALSE" } [rev['review_type']], str(rev['review_content']) [:15], str(rev['review_date']) [:10])
                  print (f "{(i+1)/total:%}\r", end='')
90
91
          # follow linking
92
           collection = dbname['user']
93
           \begin{array}{l} \text{uids} = \left[x\left['\_\text{id}'\right] \text{ for } x \text{ in } \text{list(collection.find(\{''date\_\text{of\_birth}'':\{''\$\text{exists}'': \text{True}\}\}, \{''\_\text{id}'':1\}))}\right] \\ \text{cids} = \left[x\left['\_\text{id}'\right] \text{ for } x \text{ in } \text{list(collection.find(\{''date\_\text{of\_birth}'':\{''\$\text{exists}'': \text{False}\}\}, \{''\_\text{id}'':1\}))}\right] \\ \end{array} 
94
95
           total = len(uids)
96
           print(f"\nfollow {total = }")
97
           for i, user in enumerate(uids):
98
                  shuffle (cids)
99
                  for j in range(randint(0, 20)):
100
                        graphDB.addFollow(user, cids[j])
101
                  print(f"{i/total:%}\r", end='')
102
103
           graphDB.close()
104
```

Listing 6: Test

B Mongosh scripts

B.1 Perform the escape on the string fields

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

```
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(
      x \Rightarrow \{
           print(x.primaryTitle);
          x.review.forEach(rev =>{
               if(typeof (rev.review_date) === "string" ){
                   db.movie.updateOne(
                        {primaryTitle: x.primaryTitle },
9
                        { $set: { "review.$[elem].review_date" : new Date(rev.
10
      review_date) } },
                         { arrayFilters: [ { "elem.critic_name": rev.critic name }
11
12
13
14
           print(100*i++/total);
15
16 });
```

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:
      {"review.critic_name":{ $ne: null}}}).values.length
_{2} i = 0;
3 db.runCommand(
     distinct: "movie", key: "review.critic_name", query: {"review.critic_name"
      : \{ ne: null \} \} ) . values . for Each (
       (x) \Rightarrow \{
            review arr = []
            movie_arr = []
            is\_top = false
            db.movie.aggregate(
                     { $project:
11
12
                              index: { $indexOfArray: ["$review.critic_name", x]},
13
                              primary Title: 1
14
15
                     \{ $match:\{ index: \{ $gt:-1 \} \} \}
16
17
            ).forEach(
18
19
                     tmp = db.movie.aggregate([
20
21
                               $project:
22
23
                                   top_critic: {
24
```

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

01)

Listing 9: Test

C MongoDB indexes:Movie collection

C .1 primaryTitle

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

```
git Version: 'f803681c3ae19817d31958965850193de067c516'
    },
51
    serverParameters: {
52
      internal Query Facet Buffer Size Bytes: 104857600,
53
      internalQueryFacetMaxOutputDocSizeBytes: 104857600,
54
      internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
55
      internal Document Source Group Max Memory Bytes: 104857600,
56
      internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
57
      internalQueryProhibitBlockingMergeOnMongoS: 0,
58
      internal Query Max Add To Set Bytes: 104857600,
59
      internal Document Source Set Window Fields Max Memory Bytes: 104857600
60
    },
61
    ok: 1,
62
    '$clusterTime': {
63
      clusterTime: Timestamp({ t: 1673280853, i: 1 }),
64
      signature: {
65
        hash: Binary (Buffer.from("00000000000000000000000000000000000", "
66
     hex"), 0),
        keyId: Long("0")
67
69
    operationTime: Timestamp({ t: 1673280853, i: 1 })
70
71 }
```

Listing 10: Test

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

```
totalDocsExamined: 1,
       executionStages: {
36
         stage: 'FETCH',
37
         nReturned: 1,
38
         executionTimeMillisEstimate: 0,
         works: 2,
40
         advanced: 1,
41
         needTime: 0,
42
         needYield: 0,
43
         saveState: 0,
44
         restoreState: 0,
45
         isEOF: 1,
         docsExamined: 1,
47
         alreadyHasObj: 0,
48
         inputStage: {
49
           stage: 'IXSCAN',
50
51
           nReturned: 1,
           executionTimeMillisEstimate: 0,
52
           works: 2,
53
           advanced: 1,
54
           needTime: 0,
55
           needYield: 0.
56
           saveState: 0,
57
           restoreState: 0,
58
           isEOF: 1,
59
           keyPattern: { primaryTitle: 1 },
60
           indexName: 'primaryTitle_1',
61
           isMultiKey: false,
           multiKeyPaths: { primaryTitle: [] },
63
           isUnique: false,
64
           isSparse: false,
65
           isPartial: false,
66
           index Version: 2,
67
           direction: 'forward',
68
           indexBounds: { primaryTitle: [ '["Evidence", "Evidence"]' ] },
69
70
           keysExamined: 1,
           seeks: 1,
71
           dupsTested: 0,
72
           dupsDropped: 0
73
74
75
76
    command: {
77
       find: 'movie',
78
       filter: { primaryTitle: 'Evidence' },
79
       '$db' 'rottenMovies'
80
     },
81
    serverInfo: {
82
       host: 'Profile2022LARGE10',
83
       port: 27017,
84
       version: '6.0.3',
85
       git Version: 'f803681c3ae19817d31958965850193de067c516'
86
87
    serverParameters: {
88
       internal Query Facet Buffer Size Bytes: 104857600,
89
       internal Query Facet Max Output Doc Size Bytes: 104857600,
90
       internal Look up Stage Intermediate Document Max Size Bytes: 104857600,
91
       internal Document Source Group Max Memory Bytes: 104857600,
92
       internal Query Max Blocking Sort Memory Usage Bytes: 104857600,
93
       internalQueryProhibitBlockingMergeOnMongoS: 0,
94
       internal Query MaxAdd To Set Bytes: 104857600,
95
```

```
internal Document Source Set Window Fields Max Memory Bytes: 104857600
    },
97
    ok: 1,
98
    '$clusterTime': {
99
      clusterTime: Timestamp({ t: 1673285103, i: 1 }),
100
      signature: {
101
        102
     \mathtt{hex"}), 0),
        keyId: Long("0")
103
104
105
    operationTime: Timestamp({ t: 1673285103, i: 1 })
106
107 }
```

Listing 11: Test

C.2 year

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

```
command: { find: 'movie', filter: { year: 2012 }, '$db': 'rottenMovies'
    serverInfo: {
42
      host: 'Profile2022LARGE10',
43
      port: 27017,
44
      version: '6.0.3',
45
      git Version: 'f803681c3ae19817d31958965850193de067c516'
46
47
    serverParameters: {
48
      internal Query Facet Buffer Size Bytes: 104857600,
49
      internal Query Facet Max Output Doc Size Bytes: 104857600,
50
      internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
51
      internalDocumentSourceGroupMaxMemoryBytes: 104857600,
      internal Query Max Blocking Sort Memory Usage Bytes: 104857600,
53
      internalQueryProhibitBlockingMergeOnMongoS: 0,
54
      internal Query Max Add To Set Bytes: 104857600,
55
      internal Document Source Set Window Fields Max Memory Bytes: 104857600
56
57
    },
    ok: 1,
58
    '$clusterTime': {
59
      clusterTime: Timestamp({ t: 1673280923, i: 1 }),
60
61
      signature: {
        62
     \mathtt{hex"}), 0),
        keyId: Long("0")
63
64
65
    operationTime: Timestamp({ t: 1673280923, i: 1 })
66
67
```

Listing 12: Test

```
1 {
2
    explain Version: '1',
    queryPlanner: {
3
      namespace: 'rottenMovies.movie',
4
       indexFilterSet: false,
5
       parsed Query: { year: { '$eq': 2012 } },
      queryHash: '412E8B51',
       planCacheKey: '62915BA3',
       maxIndexedOrSolutionsReached: false,
       maxIndexedAndSolutionsReached: false,
10
       maxScansToExplodeReached: false,
11
       winningPlan: {
12
         stage: 'FETCH',
13
         inputStage: {
14
           stage: 'IXSCAN',
15
           keyPattern: { year: 1 },
16
           indexName: 'year_1',
17
           isMultiKey: false,
18
           multiKeyPaths: { year: [] },
19
           isUnique: false,
20
           isSparse: false,
21
           isPartial: false,
22
           index Version: 2,
23
           direction: 'forward',
24
           indexBounds: { year: [ '[2012, 2012] ' ] }
26
27
       rejected Plans: []
28
29
```

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

```
internal Query Max Add To Set Bytes: 104857600,
     internal Document Source Set Window Fields Max Memory Bytes: 104857600
92
93
    ok: 1,
94
    '$clusterTime': {
      clusterTime: Timestamp({ t: 1673285143, i: 1 }),
96
      signature: {
97
       98
     hex"), 0),
       keyId: Long("0")
99
100
101
    operationTime: Timestamp({ t: 1673285143, i: 1 })
102
103
```

Listing 13: Test

C .3 top critic rating

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

```
type: 'simple',
         totalDataSizeSorted: 262640385,
41
         usedDisk: true,
42
         spills: 3,
43
         inputStage: {
44
           stage: 'COLLSCAN',
45
           nReturned: 14104,
46
           executionTimeMillisEstimate: 0,
47
           works: 14106,
48
           advanced: 14104,
49
           needTime: 1,
50
           needYield: 0,
51
           saveState: 47,
52
           restoreState: 47,
53
           isEOF: 1,
54
           direction: 'forward',
55
56
           docsExamined: 14104
57
58
    },
59
    command: {
60
       find: 'movie',
61
       filter: {},
62
       sort: { top critic rating: 1 },
63
       '$db': 'rottenMovies'
64
    },
65
66
    serverInfo: {
       host: 'Profile2022LARGE10',
67
       port: 27017,
68
       version: '6.0.3',
69
       git Version: 'f803681c3ae19817d31958965850193de067c516'
70
    },
71
    serverParameters: {
72
       internal Query Facet Buffer Size Bytes: 104857600,
73
       internal Query Facet Max Output Doc Size Bytes: 104857600,
74
       internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
75
      internalDocumentSourceGroupMaxMemoryBytes: 104857600,
76
       internal Query Max Blocking Sort Memory Usage Bytes: 104857600,
77
       internalQueryProhibitBlockingMergeOnMongoS: 0,
78
       internal Query Max Add To Set Bytes: 104857600,
79
      internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
80
    },
81
    ok: 1,
82
    '$clusterTime': {
83
       clusterTime: Timestamp({ t: 1673287293, i: 1 }),
84
85
       signature: {
         hash: Binary (Buffer.from("00000000000000000000000000000000000", "
86
      hex"), 0),
         keyId: Long("0")
87
88
89
    operationTime: Timestamp({ t: 1673287293, i: 1 })
90
91 }
```

Listing 14: Test

```
explainVersion: '1',
queryPlanner: {
namespace: 'rottenMovies.movie',
```

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

```
isPartial: false,
            index Version: 2,
67
            direction: 'forward',
68
            indexBounds: { top_critic_rating: [ '[MinKey, MaxKey]' ] },
69
            keysExamined: 14104,
70
            seeks: 1,
71
            dupsTested: 0,
72
            dupsDropped: 0
73
74
75
     },
76
    command: {
77
       find: 'movie',
78
       filter: \{\},
79
       sort: { top_critic_rating: 1 },
80
       '$db' 'rottenMovies'
81
82
     },
     serverInfo: {
83
       host: 'Profile2022LARGE10',
84
       port: 27017,
85
86
       version: '6.0.3',
       git Version: 'f803681c3ae19817d31958965850193de067c516'
87
88
     serverParameters: {
89
       internal Query Facet Buffer Size Bytes: 104857600,
90
       internal Query Facet Max Output Doc Size Bytes: 104857600,
91
       internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
92
       internal Document Source Group Max Memory Bytes: 104857600,
       internal Query Max Blocking Sort Memory Usage Bytes: 104857600,
94
       internalQueryProhibitBlockingMergeOnMongoS: 0,
95
       internal Query Max Add To Set Bytes: 104857600,
96
       internal Document Source Set Window Fields Max Memory Bytes: 104857600
97
98
     },
     ok: 1,
99
     '$clusterTime': {
100
       clusterTime: Timestamp({ t: 1673285423, i: 1 }),
101
       signature: {
102
         hash: Binary (Buffer.from("00000000000000000000000000000000000", "
103
      hex"), 0),
         keyId: Long("0")
104
105
106
     operationTime: Timestamp({ t: 1673285423, i: 1 })
107
108 }
```

Listing 15: Test

C .4 user rating

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

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

```
serverParameters: {
72
       internal Query Facet Buffer Size Bytes: 104857600,
73
       internal Query Facet Max Output Doc Size Bytes: 104857600,
74
       internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
75
       internal Document Source Group Max Memory Bytes: 104857600,
76
       internal Query Max Blocking Sort Memory Usage Bytes: 104857600,
77
       internalQueryProhibitBlockingMergeOnMongoS: 0,
78
       internal Query Max Add To Set Bytes: 104857600,
79
      internal Document Source Set Window Fields Max Memory Bytes: 104857600
80
    },
81
    ok: 1,
82
    '$clusterTime': {
83
       cluster Time: Timestamp (\{t: 1673287353, i: 1\}),
84
      signature: {
85
         hash: Binary (Buffer.from("00000000000000000000000000000000000", "
86
      hex"), 0),
         keyId: Long("0")
87
88
89
    operationTime: Timestamp({ t: 1673287353, i: 1 })
90
91 }
```

Listing 16: Test

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

```
executionStages: {
         stage: 'FETCH',
37
         nReturned: 14104,
38
         executionTimeMillisEstimate: 5,
39
         works: 14105,
40
         advanced: 14104,
41
         needTime: 0,
42
         needYield: 0,
43
         saveState: 14,
44
         restoreState: 14,
45
         isEOF: 1,
46
         docsExamined: 14104,
47
         alreadyHasObj: 0,
48
         inputStage: {
49
           stage: 'IXSCAN',
50
           nReturned: 14104,
51
52
           executionTimeMillisEstimate: 2,
           works: 14105,
53
           advanced: 14104,
54
           needTime: 0,
55
           needYield: 0,
56
           saveState: 14,
57
           restoreState: 14,
58
           isEOF: 1,
59
           keyPattern: { user rating: 1 },
60
           indexName: 'user_rating_1',
61
           isMultiKey: false,
62
           multiKeyPaths: { user_rating: [] },
63
           isUnique: false,
64
           isSparse: false,
65
           isPartial: false,
66
           index Version: 2,
67
           direction: 'forward',
68
           indexBounds: { user_rating: [ '[MinKey, MaxKey]' ] },
69
           keysExamined: 14104,
70
71
           seeks: 1,
           dupsTested: 0,
72
           dupsDropped: 0
73
74
75
     },
76
    command: {
77
       find: 'movie',
78
       filter: \{\},
79
       sort: { user_rating: 1 },
80
       '$db' 'rottenMovies'
81
82
     },
     serverInfo: {
83
       host: 'Profile2022LARGE10',
84
       port: 27017,
85
       version: '6.0.3',
86
       git Version: 'f803681c3ae19817d31958965850193de067c516'
87
88
     serverParameters: {
89
       internal Query Facet Buffer Size Bytes: 104857600,
90
       internal Query Facet Max Output Doc Size Bytes: 104857600,
91
       internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
92
       internal Document Source Group Max Memory Bytes: 104857600,
93
       internal Query Max Blocking Sort Memory Usage Bytes: 104857600,
94
       internalQueryProhibitBlockingMergeOnMongoS: 0,
95
       internal Query MaxAdd To Set Bytes: 104857600,
96
```

```
internal Document Source Set Window Fields Max Memory Bytes: 104857600
    },
98
    ok: 1,
99
    '$clusterTime': {
100
      clusterTime: Timestamp({ t: 1673285383, i: 1 }),
101
      signature: {
102
       103
     hex"), 0),
       keyId: Long("0")
104
105
106
    operationTime: Timestamp({ t: 1673285383, i: 1 })
107
108 }
```

Listing 17: Test

C.5 personnel.primaryName

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

```
command: {
41
      find: 'movie',
42
      filter: { 'personnel.primaryName': '' },
43
      '$db': 'rottenMovies'
44
45
    serverInfo: {
46
      host: 'Profile2022LARGE10',
47
      port: 27017,
48
      version: '6.0.3',
49
      git Version: 'f803681c3ae19817d31958965850193de067c516'
50
51
    serverParameters: {
52
      internal Query Facet Buffer Size Bytes: 104857600,
53
      internal Query Facet Max Output Doc Size Bytes: 104857600,
54
      internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
55
      internalDocumentSourceGroupMaxMemoryBytes: 104857600,
56
57
      internal Query Max Blocking Sort Memory Usage Bytes: 104857600,
      internalQueryProhibitBlockingMergeOnMongoS: 0,
58
      internal Query Max Add To Set Bytes: 104857600,
59
      internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
60
61
    ok: 1,
62
    '$clusterTime': {
63
      clusterTime: Timestamp({ t: 1673287593, i: 1 }),
64
65
      signature: {
        66
     hex"), 0),
        keyId: Long("0")
67
68
    },
69
    operationTime: Timestamp({ t: 1673287593, i: 1 })
70
71 }
```

Listing 18: Test

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

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

```
serverParameters: {
88
      internal Query Facet Buffer Size Bytes: 104857600,
89
      internal Query Facet Max Output Doc Size Bytes: 104857600,
90
      internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
91
      internal Document Source Group Max Memory Bytes: 104857600,
92
      internal Query Max Blocking Sort Memory Usage Bytes: 104857600,
93
      internalQueryProhibitBlockingMergeOnMongoS: 0,
94
      internalQueryMaxAddToSetBytes: 104857600,
      internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
96
    },
97
    ok: 1,
98
    '$clusterTime': {
99
      cluster Time: Timestamp (\{t: 1673287763, i: 1\}),
100
      signature: {
101
        102
     hex"), 0),
        keyId: Long("0")
103
104
105
    operationTime: Timestamp({ t: 1673287763, i: 1 })
106
107 }
```

Listing 19: Test

D MongoDB indexes:User collection

D.1 username

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

```
executionTimeMillisEstimate: 0,
         works: 8341,
30
         advanced: 1.
31
         needTime: 8339,
32
         needYield: 0,
33
         saveState: 8,
34
         restoreState: 8,
35
        isEOF: 1,
36
         direction: 'forward',
37
         docsExamined: 8339
38
39
    },
40
    command: {
41
       find: 'user',
42
       filter: { username: 'Abbie Bernstein' },
43
       '$db' 'rottenMovies'
44
45
    },
    serverInfo: {
46
       host: 'Profile2022LARGE10',
47
       port: 27017,
48
49
       version: '6.0.3',
      git Version: 'f803681c3ae19817d31958965850193de067c516'
50
51
    serverParameters: {
52
       internal Query Facet Buffer Size Bytes: 104857600,
53
       internalQueryFacetMaxOutputDocSizeBytes: 104857600,
54
       internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
55
       internal Document Source Group Max Memory Bytes: 104857600,
       internal Query Max Blocking Sort Memory Usage Bytes: 104857600,
57
       internalQueryProhibitBlockingMergeOnMongoS: 0,
58
      internal Query Max Add To Set Bytes: 104857600,
59
      internal Document Source Set Window Fields Max Memory Bytes: 104857600
60
61
    },
    ok: 1,
62
    '$clusterTime': {
63
       clusterTime: Timestamp({ t: 1673280753, i: 1 }),
64
      signature: {
65
         hash: Binary (Buffer.from("00000000000000000000000000000000000", "
66
      hex"), 0),
         keyId: Long("0")
67
68
69
    operationTime: Timestamp({ t: 1673280753, i: 1 })
70
71 }
```

Listing 20: Test

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

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

```
dupsDropped: 0
         }
74
       }
75
    },
76
    command: \{
77
       find: 'user',
78
       filter: { username: 'Abbie Bernstein' },
79
       '$db': 'rottenMovies'
80
     },
81
82
    serverInfo: {
       host: 'Profile2022LARGE10',
83
       port: 27017,
       version: '6.0.3',
85
       git Version: 'f803681c3ae19817d31958965850193de067c516'
86
    },
87
     serverParameters: {
88
89
       internal Query Facet Buffer Size Bytes: 104857600,
       internal Query Facet Max Output Doc Size Bytes: 104857600,
90
       internal Look up Stage Intermediate Document Max Size Bytes:\ 104857600\,,
91
       internal Document Source Group Max Memory Bytes: 104857600,
92
93
       internal Query Max Blocking Sort Memory Usage Bytes: 104857600,
       internalQueryProhibitBlockingMergeOnMongoS: 0,
94
       internal Query Max Add To Set Bytes: 104857600,
95
       internal Document Source Set Window Fields Max Memory Bytes: 104857600
96
97
     },
    ok: 1,
98
     '$clusterTime': {
99
       clusterTime: Timestamp({ t: 1673285013, i: 1 }),
       signature: {
101
         102
      hex"), 0),
         keyId: Long("0")
103
104
105
     operationTime: Timestamp({ t: 1673285013, i: 1 })
106
107 }
```

Listing 21: Test

D.2 date of birth

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

```
stage: 'COLLSCAN',
         filter: {
18
           date of birth: {
19
              '$eq': 'Mon Jan 09 2023 17:05:07 GMT+0000 (Western European
20
      Standard Time)'
21
22
         direction: 'forward'
23
       },
24
       rejected Plans: []
25
26
     executionStats: {
27
       executionSuccess: true,
28
       nReturned: 0,
29
       executionTimeMillis: 32,
30
       totalKeysExamined: 0,
31
32
       totalDocsExamined: 8339,
       executionStages: {
33
         stage: 'COLLSCAN',
34
         filter: {
35
           date_of_birth: {
36
              '$eq': 'Mon Jan 09 2023 17:05:07 GMT+0000 (Western European
37
      Standard Time),
           }
38
         },
39
         nReturned: 0,
40
         executionTimeMillisEstimate: 23,
41
         works: 8341,
42
         advanced: 0,
43
         needTime: 8340,
44
         needYield: 0,
45
         saveState: 8,
46
         restoreState: 8,
47
         isEOF: 1,
48
         direction: 'forward',
49
         docsExamined: 8339
50
51
52
    command: {
53
       find: 'user',
54
       filter: {
55
         date of birth: 'Mon Jan 09 2023 17:05:07 GMT+0000 (Western European
56
      Standard Time),
57
       '$db': 'rottenMovies'
58
59
     },
     serverInfo: {
       host: 'Profile2022LARGE10',
61
       port: 27017,
62
       version: '6.0.3',
63
       git Version: \  \, \textbf{'f803681c3ae19817d31958965850193de067c516'} \\
64
65
    serverParameters: {
66
       internal Query Facet Buffer Size Bytes: 104857600,
67
       internal Query Facet Max Output Doc Size Bytes: 104857600,
68
       internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
69
       internal Document Source Group Max Memory Bytes: \ 104857600\,,
70
       internal Query Max Blocking Sort Memory Usage Bytes: 104857600,
71
       internalQueryProhibitBlockingMergeOnMongoS: 0,
72
       internal Query Max Add To Set Bytes: 104857600,
73
       internal Document Source Set Window Fields Max Memory Bytes: 104857600
74
```

```
ok: 1,
76
    '$clusterTime': {
77
      clusterTime: Timestamp({ t: 1673283903, i: 1 }),
78
      signature: {
        hash: Binary (Buffer.from("00000000000000000000000000000000000", "
80
     hex"), 0),
        keyId: Long("0")
81
82
    },
83
    operationTime: Timestamp({ t: 1673283903, i: 1 })
84
85 }
```

Listing 22: Test

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

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

```
serverParameters: {
102
       internal Query Facet Buffer Size Bytes: 104857600,
103
       internalQueryFacetMaxOutputDocSizeBytes: 104857600,
104
       internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
105
       internal Document Source Group Max Memory Bytes: 104857600,
106
       internal Query Max Blocking Sort Memory Usage Bytes: 104857600,
107
       internalQueryProhibitBlockingMergeOnMongoS: 0,
108
       internalQueryMaxAddToSetBytes: 104857600,
109
       internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
110
     },
111
112
     ok: 1,
     '$clusterTime': {
113
       cluster Time: Timestamp (\{t: 1673285073, i: 1\}),
114
       signature: {
115
         hash: Binary (Buffer.from("00000000000000000000000000000000000", "
116
      hex"), 0),
         keyId: Long("0")
117
118
119
120
     operationTime: Timestamp({ t: 1673285073, i: 1 })
121 }
```

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"
     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>
5
        <groupId>org.springframework.boot
6
        <artifactId>spring-boot-starter-parent</artifactId>
        <version> 3.0.0</version>
        <relativePath/> <!— lookup parent from repository —>
9
     10
     <groupId>it . unipi . dii . lsmsdb/groupId>
11
     <artifactId>rottenMovies</artifactId>
12
     < version>0.0.1 -SNAPSHOT</ version>
13
     <name>rottenMovies</name>
14
     <description>Project for the rotten movies service</description>
15
     properties>
16
        <java . egin{array}{c} <java . egin{array}{c}  ersion<math>>
17
     properties>
18
     <dependencies>
19
        <dependency>
20
           <groupId>org.springframework.boot
21
           <artifactId>spring-boot-starter-thymeleaf</artifactId>
22
        </dependency>
23
        <dependency>
24
           <groupId>org.springframework.boot
25
           <artifactId>spring-boot-starter-web</artifactId>
26
        </dependency>
```

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

Listing 24: Test