ASSIGNMENT 4

Abhishek Shah, as553

Question 1

Preprocessing Assignment2 files

Update Title

Set title =replace(title,'"',' ')

Update Title

Set originalTitle= replace(originalTitle,'"','')

Update Member

Set name= replace(name,'"', '')

Creating JSON files

```
import psycopg2

def loadData():
    conn = None
    try:
        conn = psycopg2.connect(
            host='localhost',
            database='imdb6202',
            user='postgres',
            password='Abhishek@123',
            port=5432)
        cur = conn.cursor()
        loadMovies = """
        COPY (SELECT json_strip_nulls(row_to_json(res))
       FROM
      (SELECT id as _id, type, title, originalTitle, startYear, endYear,
       runtime, avgRating, numVotes, genres, actors, director as directors,
       producer as producers, writer as writers
       FROM Title

       LEFT JOIN
       (
       SELECT title as genreTitle, array_agg(Genre.genre) as genres
         FROM Title_Genre JOIN Genre
        ON Title_Genre.genre = Genre.id
         GROUP BY genreTitle
       ORDER BY genreTitle
       ) as g
       ON Title.id = g.genreTitle

       LEFT JOIN
       (
       SELECT title as directorTitle, array_agg(director) as director
         FROM Title_Director GROUP BY directorTitle) as d
       ON Title.id = d.directorTitle

       LEFT JOIN
       (
       SELECT title as producerTitle, array_agg(producer) as producer
         FROM Title_Producer GROUP BY producerTitle) as p
       ON Title.id = p.producerTitle

       LEFT JOIN
       (
       SELECT title as writerTitle, array_agg(writer) as writer
         FROM Title_Writer GROUP BY writerTitle) as w
       ON Title.id = w.writerTitle

       LEFT JOIN
       (
       SELECT atcTitle, json_agg(actors) as actors FROM
         (
       SELECT ta.atcTitle, json_build_object('actor', ta.actorid, 'roles', ta.roles)
as actors FROM
         (
       SELECT atc.title as atcTitle, atc.actor as actorid,
array_agg(character.character) as roles
         FROM character JOIN actor_title_character atc
        ON character.id = atc.character
         GROUP BY (atc.title, atc.actor)
       ORDER BY (atc.title, atc.actor)) as ta) AS result
         GROUP BY atcTitle) as t ON t.atcTitle = Title.id
       ) res)
        TO '/Users/abhishekshah/Downloads/Movies.json' with (FORMAT TEXT, HEADER
false);
        """
        cur.execute(loadMovies)
        print("Rows affected for Movies: ", cur.rowcount)

      loadMembers = """
        COPY (
       SELECT json_strip_nulls(row_to_json(res))
       FROM(
      SELECT id as _id, name, birthyear, deathyear
      FROM Member
        )res)
        TO '/Users/abhishekshah/Downloads/Test2.json' with (FORMAT TEXT, HEADER
false);
        """
        cur.execute(loadMembers)
        print("Rows affected for Members: ", cur.rowcount)
        cur.close()

      except(Exception, psycopg2.DatabaseError) as error:
        print(error)
      finally:
        if conn is not None:
            conn.close()
            print("Connection closed")


if __name__ == "__main__":
    loadData()
```

Loading Data

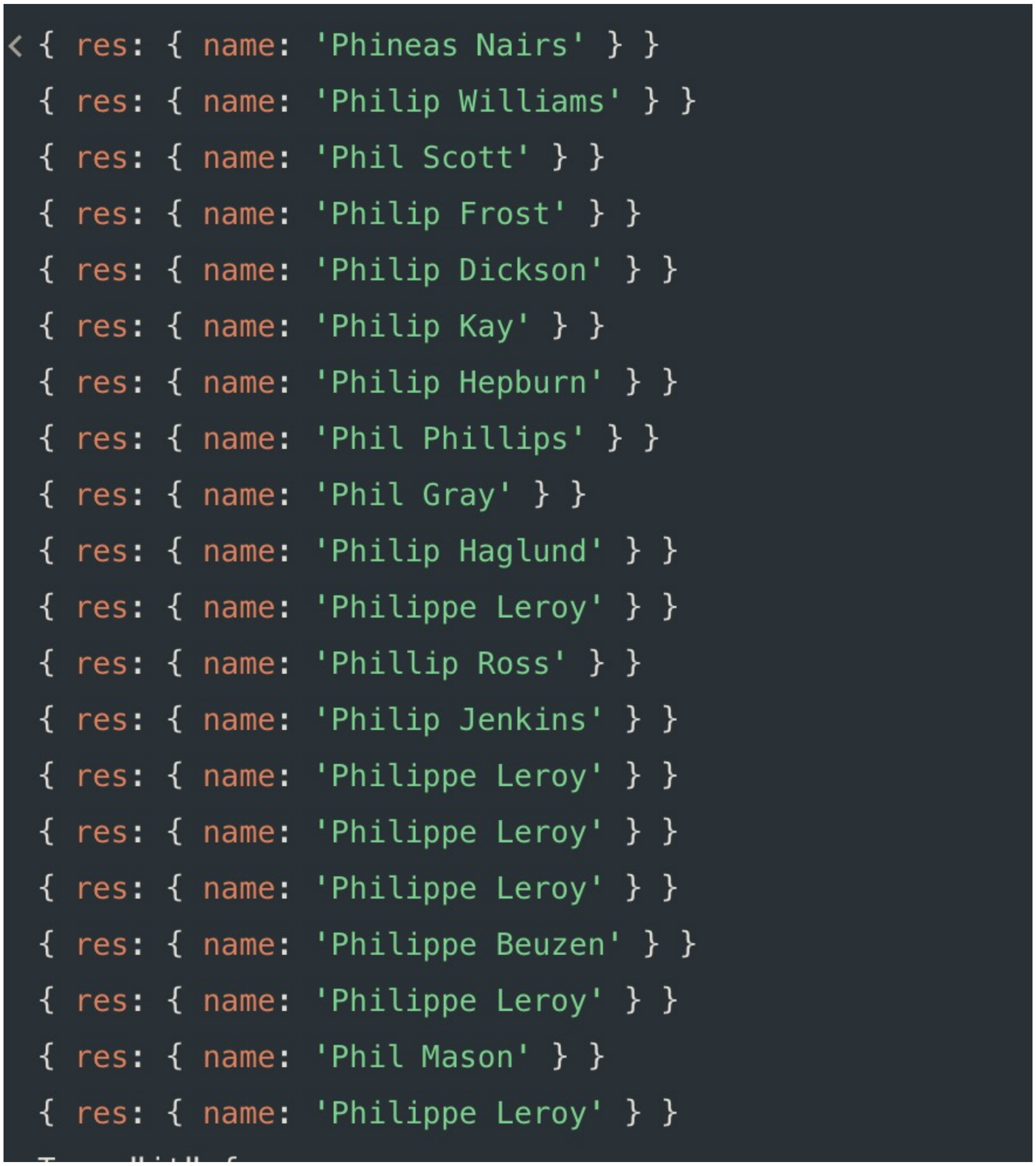
```
mongoimport --db imdb620 --collection Movies --file
/Users/abhishekshah/Downloads/Movies.json

mongoimport --db imdb620 --collection Members --file
/Users/abhishekshah/Downloads/Members.json
```

Question 2

2.1

```
db.Movies.aggregate([
  {
    $match: {"startyear": {$ne:2014}}},
  {
    $lookup:
  {
    from: "Members",
    localField: "actors.actor",
    foreignField: "_id",
    as: "res"}},
  {
    $unwind: "$res"},
  {
    $match:{"res.name": /^Phi/, "res.deathyear": null}},
  {
    $project:{"_id":0,"res.name":1}}
    ]);
-- 5sec 806msec
```

<u>Output</u>



```
< { res: { name: 'Phineas Nairs' } }
  { res: { name: 'Philip Williams' } }
  { res: { name: 'Phil Scott' } }
  { res: { name: 'Philip Frost' } }
  { res: { name: 'Philip Dickson' } }
  { res: { name: 'Philip Kay' } }
  { res: { name: 'Philip Hepburn' } }
  { res: { name: 'Phil Phillips' } }
  { res: { name: 'Phil Gray' } }
  { res: { name: 'Philip Haglund' } }
  { res: { name: 'Philippe Leroy' } }
  { res: { name: 'Phillip Ross' } }
  { res: { name: 'Philip Jenkins' } }
  { res: { name: 'Philippe Leroy' } }
  { res: { name: 'Philippe Leroy' } }
  { res: { name: 'Philippe Leroy' } }
  { res: { name: 'Philippe Beuzen' } }
  { res: { name: 'Philippe Leroy' } }
  { res: { name: 'Phil Mason' } }
  { res: { name: 'Philippe Leroy' } }
```

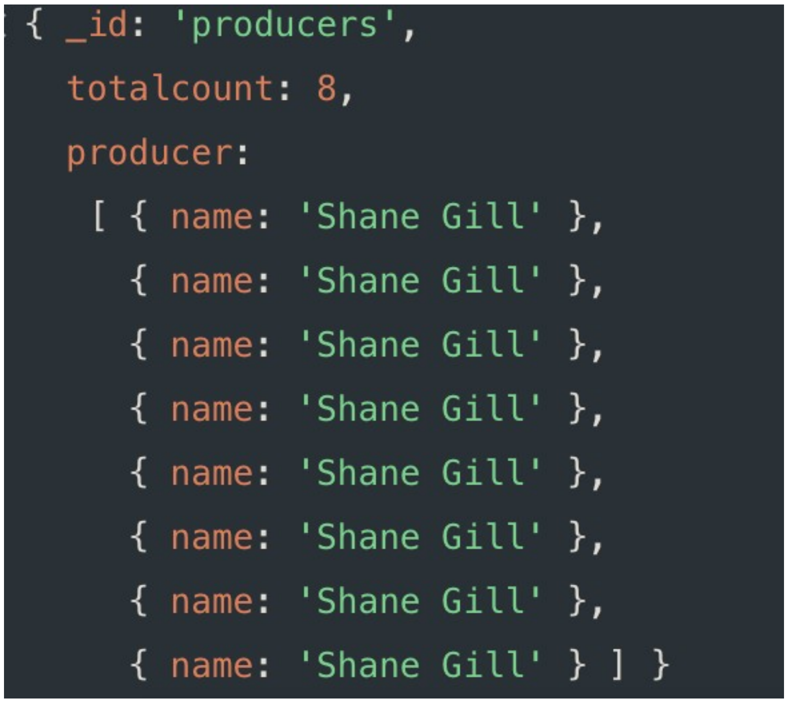
2.2

```
db.Movies.aggregate([{
 $match:{"startyear": {$eq:2017}, "genres": {$eq: "Talk-Show"}}},
 {
 $lookup:
 {
```

```
from: "Members",
localField:"producers",
foreignField:"_id",
as:"new"}},
{
  $unwind: "$new"},
{
  $match:{"new.name":{"$regex":/Gill/}}},
{
  $group: { _id: "producers", totalcount: {"$sum":1},
  producer:{$push:{name:"$new.name"}}}
}
]);
```

-- 3sec 652msec

Output



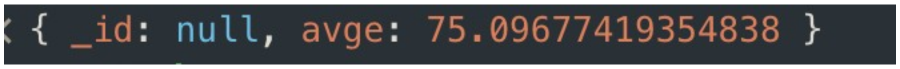
```
{ _id: 'producers',
  totalcount: 8,
  producer:
   [ { name: 'Shane Gill' },
     { name: 'Shane Gill' },
     { name: 'Shane Gill' },
     { name: 'Shane Gill' },
     { name: 'Shane Gill' },
     { name: 'Shane Gill' },
     { name: 'Shane Gill' },
     { name: 'Shane Gill' } ] }
```

2.3

```
db.Movies.aggregate([
  {
    $lookup:
  {
    from: "Members",
    localField: "writers",
    foreignField: "_id",
    as: "res"}},
  {
    $match: {"res.name": /Bhardwaj/, "res.deathyear": null}},
  {
    $group: {"_id": null, "average": {"$avg": "$runtime"}}}
  ]);
```

-- 1min 33sec

Output



```
{ _id: null, avge: 75.09677419354838 }
```

2.4

db.Movies.aggregate([

{

 $match: {"runtime": {$gt: 120}}},

 {

 $lookup:

 {

 from: "Members",

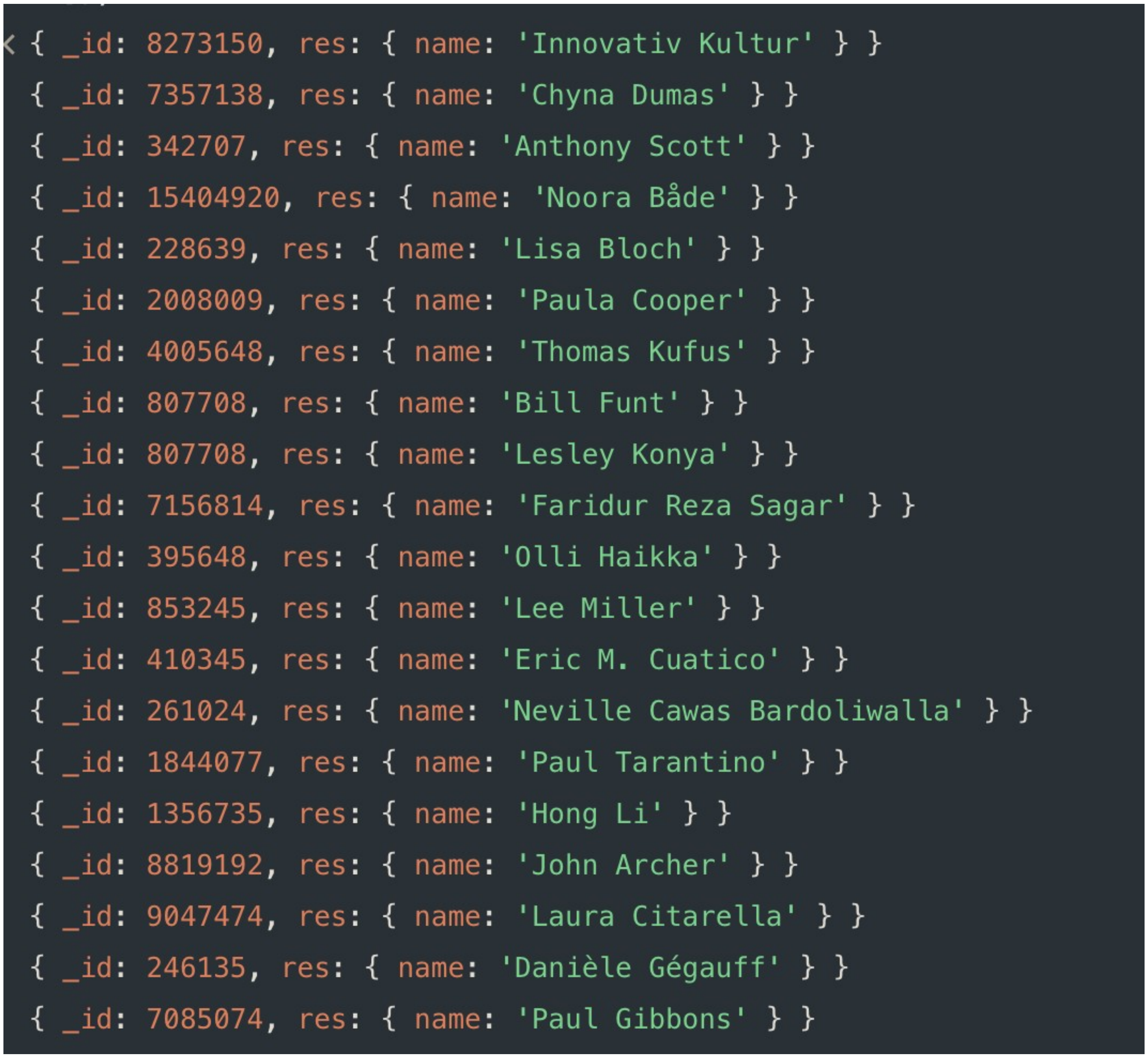
 localField: "producers",

 foreignField: "_id",

 as: "res"}},

```
 {
 $unwind: "$res"},
 {
 $match:{"res.deathyear": null}},
 {
 $sort:{"runtime": -1}},
 {
 $project:{"_id": 1,"res.name": 1}}
 ]);
-- 3 sec 804sec
```

Output



```
< { _id: 8273150, res: { name: 'Innovativ Kultur' } }
  { _id: 7357138, res: { name: 'Chyna Dumas' } }
  { _id: 342707, res: { name: 'Anthony Scott' } }
  { _id: 15404920, res: { name: 'Noora Både' } }
  { _id: 228639, res: { name: 'Lisa Bloch' } }
  { _id: 2008009, res: { name: 'Paula Cooper' } }
  { _id: 4005648, res: { name: 'Thomas Kufus' } }
  { _id: 807708, res: { name: 'Bill Funt' } }
  { _id: 807708, res: { name: 'Lesley Konya' } }
  { _id: 7156814, res: { name: 'Faridur Reza Sagar' } }
  { _id: 395648, res: { name: 'Olli Haikka' } }
  { _id: 853245, res: { name: 'Lee Miller' } }
  { _id: 410345, res: { name: 'Eric M. Cuatico' } }
  { _id: 261024, res: { name: 'Neville Cawas Bardoliwalla' } }
  { _id: 1844077, res: { name: 'Paul Tarantino' } }
  { _id: 1356735, res: { name: 'Hong Li' } }
  { _id: 8819192, res: { name: 'John Archer' } }
  { _id: 9047474, res: { name: 'Laura Citarella' } }
  { _id: 246135, res: { name: 'Danièle Gégauff' } }
  { _id: 7085074, res: { name: 'Paul Gibbons' } }
```

2.5

```
db.Movies.aggregate([
  {
  $match:{"genres": {"$eq": "Sci-Fi"}}},
  {
  $unwind: "$directors"},
  {
  $lookup:
    {
    from: "Members",
    localField: "directors",
    foreignField: "_id",
    as: "result"}
    },
         {
         $match:{"result.name": {"$eq": "James Cameron"}}},
         {
         $unwind: "$actors"},
    {
    $lookup:
    {
    from: "Members",
    localField: "actors.actor",
    foreignField: "_id",
    as: "result1"}
    },
    {
    $match:{"result1.name": {"$eq": "Sigourney Weaver"}}},
    {
    $project:{_id: 1, "title": 1},
  }
]);
```

<u>Output</u>

I think, my logic is correct but I am unable to get any output.

I think, my data from assignment 2 only contains actors and not actresses.

Question 3

3.1

Explain.queryPlanner

'namespace' is the name of database followed by collection on which the query is executed, which is imdb620.Movies where imdb620 is the database and Movies is collection.

'queryHash' is use to hash query and is primarily used for the slow queries and the possible of hash collisions and is very useful.

'maxIndexedOrSolutionsReached' 'maxIndexedAndSolutionsReached' is nothing but a boolean value if the maximum buffer size is used.

Explain.queryPlanner.winningPlan

'winningPlan' is a document that details the plan selected by the query optimizer which is 'COLLSCAN' in our case.

'rejectedPlans' is none in this particular query.

Explain.ExecutionStats

This particular section provides various type of statistical data related to our query.

'executionSuccess' is a boolean status if our query was run properly.

'executionTimeMillisEstimate' : **159**

'docsExamined' refers to the total number of documents examined and not the number of documents returned: **1189930**

'nReturned' is the number of documents that match the given query condition: **1143166**

Explain.ExecutionStats.ExecutionStages

'executionTimeMillisEstimate' is the estimated amount of time in milliseconds for query execution : **81750**

'works' specifies the number of work units performed by the query execution stage: **1189932**

'saveState' shows the number of times the query stopped and saved the current state: **1242**

'needTime' is the number of work cycles that did not advance an intermediate result to its parent stage: **46765**

'needYield' is the number of times that the storage layer requested that the query stage suspend processing and yield its locks: **0** 'restoreState' similar to above shows how many times it restored the current state: **1248**

'isEOF' specifies whether the execution stage has reached end of stream or not.

The explanation for lookup is as follows

explain.totalDocsExamined: **2234655**

explain.totalKeysExamined: **4638350**

explain.collectionScans: **0**

explain.indexesUsed: **[id]** (default)

explain.nReturned: **7901**

explain.executionTimeMillisEstimate: **102278**

SERVERINFO and SERVERPARAMETERS

Contain the information about the name, port of the host.
It contains information about the maximum buffer size for different stages of the execution plan.
Which is **104857600** bytes.

In the end, just the **command/query** we ran is returned.

3.2

Explain.queryPlanner

'namespace' is the name of database followed by collection on which the query is executed, which is imdb620.Movies where imdb620 is the database and Movies is collection.

'queryHash' is use to hash query and is primarily used for the slow queries and the possible of hash collisions and is very useful.

'maxIndexedOrSolutionsReached' 'maxIndexedAndSolutionsReached' is nothing but a boolean value if the maximum buffer size is used.

Explain.queryPlanner.winningPlan

'winningPlan' is a document that details the plan selected by the query optimizer which is 'COLLSCAN' in our case.

'rejectedPlans' is none in this particular query.

Explain.ExecutionStats

This particular section provides various type of statistical data related to our query.

'executionSuccess' is a boolean status if our query was run properly.

'executionTimeMillisEstimate' : **71**

'docsExamined' refers to the total number of documents examined and not the number of documents returned: **1189930**

'nReturned' is the number of documents that match the given query condition: **2876**

Explain.ExecutionStats.ExecutionStages

'executionTimeMillisEstimate' is the estimated amount of time in milliseconds for query execution : **1313**

'works' specifies the number of work units performed by the query execution stage: **1189932**

'saveState' shows the number of times the query stopped and saved the current state: **1190**

'needTime' is the number of work cycles that did not advance an intermediate result to its parent stage: **1187055**

'needYield' is the number of times that the storage layer requested that the query stage suspend processing and yield its locks: **0**

'restoreState' similar to above shows how many times it restored the current state: **1190**

'isEOF' specifies whether the execution stage has reached end of stream or not.

The explanation for lookup is as follows

explain.totalDocsExamined: **574**

explain.totalKeysExamined: **3259**

explain.collectionScans: **0**

explain.indexesUsed: **[id]** (default)

explain.nReturned: **8**

explain.executionTimeMillisEstimate: **1310**

SERVERINFO and SERVERPARAMETERS

Contain the information about the name, port of the host.

It contains information about the maximum buffer size for different stages of the execution plan.

Which is **104857600** bytes.

In the end, just the **command/query** we ran is returned.

3.3

Explain.queryPlanner

'namespace' is the name of database followed by collection on which the query is executed, which is imdb620.Movies where imdb620 is the database and Movies is collection.

'queryHash' is use to hash query and is primarily used for the slow queries and the possible of hash collisions and is very useful.

'maxIndexedOrSolutionsReached' 'maxIndexedAndSolutionsReached' is nothing but a boolean value if the maximum buffer size is used.

Explain.queryPlanner.winningPlan

'winningPlan' is a document that details the plan selected by the query optimizer which is 'PROJECTION_DEFAULT' in our case.

'rejectedPlans' is none in this particular query.

Explain.ExecutionStats

This particular section provides various type of statistical data related to our query.

'executionSuccess' is a boolean status if our query was run properly.

'executionTimeMillisEstimate' : **83**

'docsExamined' refers to the total number of documents examined and not the number of documents returned: **1189930**

'nReturned' is the number of documents that match the given query condition: **1189930**

Explain.ExecutionStats.ExecutionStages

'executionTimeMillisEstimate' is the estimated amount of time in milliseconds for query execution : **1344**

'works' specifies the number of work units performed by the query execution stage: **1189932**

'saveState' shows the number of times the query stopped and saved the current state: **1242**

'needTime' is the number of work cycles that did not advance an intermediate result to its parent stage: **1**

'needYield' is the number of times that the storage layer requested that the query stage suspend processing and yield its locks: **0**

'restoreState' similar to above shows how many times it restored the current state: **1242**

'isEOF' specifies whether the execution stage has reached end of stream or not.

The explanation for lookup is as follows

explain.totalDocsExamined: **1506497**

explain.totalKeysExamined: **3264593**

explain.collectionScans: **0**

explain.indexesUsed: **[id]** (default)

explain.nReturned: **1189930**

explain.executionTimeMillisEstimate: **66350**

SERVERINFO and SERVERPARAMETERS

Contain the information about the name, port of the host.

It contains information about the maximum buffer size for different stages of the execution plan.

Which is **104857600** bytes for most of them.

In the end, just the **command/query** we ran is returned.

3.4

Explain.queryPlanner

'namespace' is the name of database followed by collection on which the query is executed, which is imdb620.Movies where imdb620 is the database and Movies is collection.

'queryHash' is use to hash query and is primarily used for the slow queries and the possible of hash collisions and is very useful.

'maxIndexedOrSolutionsReached' 'maxIndexedAndSolutionsReached' is nothing but a boolean value if the maximum buffer size is used.

Explain.queryPlanner.winningPlan

'winningPlan' is a document that details the plan selected by the query optimizer which is 'COLLSCAN' in our case.

'rejectedPlans' is none in this particular query.

Explain.ExecutionStats

This particular section provides various type of statistical data related to our query.

'executionSuccess' is a boolean status if our query was run properly.

'executionTimeMillisEstimate' : **38**

'docsExamined' refers to the total number of documents examined and not the number of documents returned: **1189930**

'nReturned' is the number of documents that match the given query condition: **36221**

Explain.ExecutionStats.ExecutionStages

'executionTimeMillisEstimate' is the estimated amount of time in milliseconds for query execution : **829**

'works' specifies the number of work units performed by the query execution stage: **1189932**

'saveState' shows the number of times the query stopped and saved the current state: **1192**

'needTime' is the number of work cycles that did not advance an intermediate result to its parent stage: **1153710** 'needYield' is the number of times that the storage layer requested that the query stage suspend processing and yield its locks: **0**

'restoreState' similar to above shows how many times it restored the current state: **1192**

'isEOF' specifies whether the execution stage has reached end of stream or not.

The explanation for lookup is as follows

explain.totalDocsExamined: **26869**

explain.totalKeysExamined: **59697**

explain.collectionScans: **0**

explain.indexesUsed: **[id]** (default)

explain.nReturned: **24331**

explain.executionTimeMillisEstimate: **3361**

SERVERINFO and SERVERPARAMETERS

Contain the information about the name, port of the host.

It contains information about the maximum buffer size for different stages of the execution plan.

Which is **104857600** bytes for most of them.

In the end, just the **command/query** we ran is returned.

3.5

Explain.queryPlanner

'namespace' is the name of database followed by collection on which the query is executed, which is imdb620.Movies where imdb620 is the database and Movies is collection.

'queryHash' is use to hash query and is primarily used for the slow queries and the possible of hash collisions and is very useful.

'maxIndexedOrSolutionsReached' 'maxIndexedAndSolutionsReached' is nothing but a boolean value if the maximum buffer size is used.

Explain.queryPlanner.winningPlan

'winningPlan' is a document that details the plan selected by the query optimizer which is 'COLLSCAN' in our case.

'rejectedPlans' is none in this particular query.

Explain.ExecutionStats

This particular section provides various type of statistical data related to our query.

'executionSuccess' is a boolean status if our query was run properly.

'executionTimeMillisEstimate' : **39**

'docsExamined' refers to the total number of documents examined and not the number of documents returned: **1189930**

'nReturned' is the number of documents that match the given query condition: **32278**

Explain.ExecutionStats.ExecutionStages

'executionTimeMillisEstimate' is the estimated amount of time in milliseconds for query execution : **37**

'works' specifies the number of work units performed by the query execution stage: **1189932**

'saveState' shows the number of times the query stopped and saved the current state: **1193**

'needTime' is the number of work cycles that did not advance an intermediate result to its parent stage: **1157653**

'needYield' is the number of times that the storage layer requested that the query stage suspend processing and yield its locks: **0**

'restoreState' similar to above shows how many times it restored the current state: **1193**

'isEOF' specifies whether the execution stage has reached end of stream or not.

The explanation for lookup is as follows

explain.totalDocsExamined: **28242**

explain.totalKeysExamined: **28242**

explain.collectionScans: **0**

explain.indexesUsed: **[id]** (default)

explain.nReturned: **28242**

explain.executionTimeMillisEstimate: **2486**

SERVERINFO and SERVERPARAMETERS

Contain the information about the name, port of the host.

It contains information about the maximum buffer size for different stages of the execution plan.

Which is **104857600.**

In the end, just the **command/query** we ran is returned.

Question 4

-- First run all the queries without creating index.
-- Indexes are special data structures that store a small portion of the collection's data set in an easy to traverse form.
-- The index stores the value of a specific field or set of fields, ordered by the value of the field.
-- The ordering of the index entries supports efficient equality matches and range-based query operations

-- In case of Members collection, we create index for columns, name and deathyear as they are frequently used retrieval.
-- In case of Movies collection, we create index for runtime, startyear as they are used frequently for data retrieval.

Creating Indexes

```
db.Members.createIndex(
  { name: 1}
)
```

```
db.Members.createIndex(
  { name: 1, deathYear: -1}
)

db.Movies.createIndex(
  { runtime: 1}
  )

db.Movies.createIndex(
{genre: -1}
)

db.Movies.createIndex(
  { startyear: -1}
)
```

Runtime

4.1

```
db.Movies.aggregate([
  {
    $match: {"startyear": {$ne:2014}}},
  {
    $lookup:
  {
    from: "Members",
    localField: "actors.actor",
    foreignField: "_id",
    as: "res"}},
  {
    $unwind: "$res"},
  {
    $match:{"res.name": /^Phi/, "res.deathyear": null}},
  {
    $project:{"_id":0,"res.name":1}}
    ]);
```

Time taken before indexing: 5sec 806msec

Tim taken after indexing: 2 sec 219msec

4.2

```
db.Movies.aggregate([{
  $match:{"startyear": {$eq:2017}, "genres": {$eq: "Talk-Show"}}},
  {
  $lookup:
  {
  from: "Members",
  localField:"producers",
  foreignField:"_id",
  as:"new"}},
  {
   $unwind: "$new"},
  {
   $match:{"new.name":{"$regex":/Gill/}}},
  {
   $group: { _id: "producers", totalcount: {"$sum":1},
   producer:{$push:{name:"$new.name"}}}
  }
  ]);
```

Time taken before indexing: 3sec 652msec

Tim taken after indexing: 2sec 321msec

4.3

```
db.Movies.aggregate([
  {
    $lookup:
  {
    from: "Members",
    localField: "writers",
    foreignField: "_id",
    as: "res"}},
  {
    $match: {"res.name": /Bhardwaj/, "res.deathyear": null}},
  {
    $group: {"_id": null, "average": {"$avg": "$runtime"}}}
  ]);
```

Time taken before indexing: 1min 33sec

Tim taken after indexing: 1min 8sec

4.4

```
db.Movies.aggregate([
{
 $match: {"runtime": {$gt: 120}}},
 {
 $lookup:
 {
 from: "Members",
 localField: "producers",
 foreignField: "_id",
 as: "res"}},
 {
 $unwind: "$res"},
 {
 $match:{"res.deathyear": null}},
 {
 $sort:{"runtime": -1}},
 {
 $project:{"_id": 1,"res.name": 1}}
 ]);
```

Time taken before indexing: 3 sec 804msec

Tim taken after indexing: 3 sec 215msec

4.5

```
db.Movies.aggregate([
  {
  $match:{"genres": {"$eq": "Sci-Fi"}}},
  {
  $unwind: "$directors"},
  {
  $lookup:
    {
    from: "Members",
    localField: "directors",
    foreignField: "_id",
    as: "result"}
    },
           {
           $match:{"result.name": {"$eq": "James Cameron"}}},
           {
           $unwind: "$actors"},
    {
    $lookup:
    {
    from: "Members",
    localField: "actors.actor",
    foreignField: "_id",
    as: "result1"}
    },


    {
    $match:{"result1.name": {"$eq": "Sigourney Weaver"}}},
    {
    $project:{_id: 1, "title": 1},
  }
]);
```

Time taken before indexing:

Tim taken after indexing: