DATABASE MANAGEMENT AND PROGRAMMING NOSQL

1.0 INTRODUCTION

This is an individual assignment, which gives you an opportunity to demonstrate your knowledge of NoSQL and your ability to implement, query and design a MongoDB document database. You will be awarded marks for what is achieved. This assignment is worth 50% of the overall module mark.

2.0 SCENARIO

Goodreads is an American social cataloguing website and a subsidiary of Amazon that allows individuals to search its database of books, annotations, quotes, and reviews. Users can sign up and register books to generate library catalogues and reading lists. They can also create their own groups of book suggestions, surveys, polls, blogs, and discussions (copied directly from the Wikipedia page).

- Goodreads has a web page for each book e.g. https://www.goodreads.com/book/show/5333265
- or each review e.g., https://www.goodreads.com/review/show/2410025795

A subset of Goodreads data has been selected for this coursework. There is data on books, reviews, authors, and genre. Permission was granted to use the data for this coursework from https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/home on condition that the authors' research papers are referenced (Mengting W. and Julian J. M 2018), (Mengting W *et al.* 2019). It is not necessary to read these papers to complete this coursework.

3.0 CONNECT, EXTRACT, TRANSFORM AND LOAD DATA (CETL) [15 marks]

3.1 Connecting to DMU MongoDB server.

Connect to MongoDB server using MongoDB Compass with the following credentials. Please note that the username and password are in lowercase.

Host	#####
Username	#####
Password	#####
Authentication Database	#####

If you are having difficulties with the credentials above, paste the below URI in the Connection String window

##

Note: *At the moment, you can only connect to the MongoDB server while on university network. This means either you are on the University lab machines or you connect via Horizon. If you wish to work on your PC, you have to complete the ETL using above methods then move to your PC and import the personalized datasets.*

3.2 Understanding the data.

Study all the collections (read more here: https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/home) and answer the following:

1. What is/are the unique/identifying key(s) of the documents of each collection? [4 marks]

   Unique/Identifying Keys:

   - Author / authors.json: The unique/identifying key for each document is the _id field.
   - Genre / genre.json: The unique/identifying key for each document is the _id field.
   - Book / books.json: The unique/identifying key for each document is the _id field.
   - Review / reviews.json: The unique/identifying key for each document is the _id field.

2. Study the collections briefly with their fields and values. [6 marks]

   a. Identify issues and anomalies you have seen such as invalid data, empty fields, etc.?

   b. Identify examples and how you intend to address them. You do not need to solve the issues here.

Complete the table below based on 2(a) and 2(b).

Collection	Field	Anomaly	Examples	Solution Plan
Author	average_rating	Invalid data type (should be numeric)	"4.06"	Convert the field to a numeric data type
Author	text_reviews_count	Invalid data type (should be numeric)	"8642"	Convert the field to a numeric data type
Author	ratings_count	Invalid data type (should be numeric)	"96780"	Convert the field to a numeric data type
Book	average_rating	Invalid data type (should be numeric)	"3.83"	Convert the field to a numeric data type
Book	is_ebook	Invalid data type (should be boolean)	"false"	Convert the field to a boolean data type

Book	num_pages	Invalid data type (should be numeric)	"128"	Convert the field to a numeric data type
Book	publication_day	Invalid data type (should be numeric)	"20"	Convert the field to a numeric data type
Book	publication_month	Invalid data type (should be numeric)	"4"	Convert the field to a numeric data type
Book	publication_year	Invalid data type (should be numeric)	"2015"	Convert the field to a numeric data type
Book	ratings_count	Invalid data type (should be numeric)	"37"	Convert the field to a numeric data type
Review	rating	Invalid data type (should be numeric)	5	Convert the field to a numeric data type
Review	n_votes	Invalid data type (should be numeric)	16	Convert the field to a numeric data type
Review	n_comments	Invalid data type (should be numeric)	0	Convert the field to a numeric data type
All	Empty fields	Missing or empty values in fields	N/A	Identify missing fields and handle them based on their required data type and relevance

All	Inconsistent data types	Data types that do not match the expected format	N/A	Convert the data types to match the expected format, such as converting strings to numbers, etc.

3.3 Getting your personalized data. [5 marks]

You are assigned a specific dataset to work on the labs. This means, the results you will get from your queries will be different from other students. Check blackboard under **My Grades -> CW Assigned Group.**

1. Extract all books matching your assignedGroup from books collection into a new collection named pxxxxxxx_books where pxxxxxxx is replaced with your p-number. [2 mark]

2. Extract all matching reviews of books in pxxxxxxx_books to a collection named pxxxxxxx_reviews. [1 mark]

   Query:

```
//3.3.2
db.review.aggregate([
  {
    $lookup: {
      from: "book",
      localField: "book_id",
      foreignField: "book_id",
      as: "matching_books"
    }
  },
  {
    $match: {
      matching_books: { $ne: [] }
    }
  },
  {
    $out: "pxxxxxxx_review"
  }
])
```

3. Extract all matching authors of books in pxxxxxxx_books to a collection named pxxxxxxx_authors. [1 mark]

   Query:

```
//3.3.3
```

```
db.author.aggregate([
  {
    $lookup: {
      from: "book",
      localField: "author_id",
      foreignField: "author_id",
      as: "matching_authors"
    }
  },
  {
    $match: {
      matching_authors: { $ne: [] }
    }
  },
  {
    $out: "pxxxxxxx_author"
  }
])
```

4. Extract all matching genres of books in pxxxxxxx_books to a new collection named pxxxxxxx_genres. [1 mark]

```
//3.3.4
db.genre.aggregate([
  {
    $lookup: {
      from: "book",
      localField: "book_id",
      foreignField: "book_id",
      as: "matching_genres"
    }
  },
  {
    $match: {
      matching_genres: { $ne: [] }
    }
  },
  {
    $out: "pxxxxxxx_genres"
  }
])
```

Export all your collections to your local machine. Then delete all newly created collections matching your p-number only. Be careful not to delete other students' collections. Disconnect from the DMU MongoDB server and connect to localhost server. For those that would want to continue the assessment on their PC, this is the time to move back to your PC!

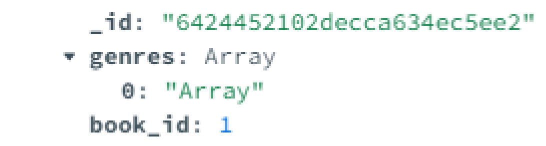
4.0 CLEANING THE COLLECTIONS [20 marks]

4.1 Have a quick look at the collections on MongoDB Compass, you will realise that some of the data was only scrapped from the internet and incomplete or not in proper format. Remember, in Section 3.2, you identified some anomalies on the dataset. For each anomaly that happens to be in your personalized dataset:

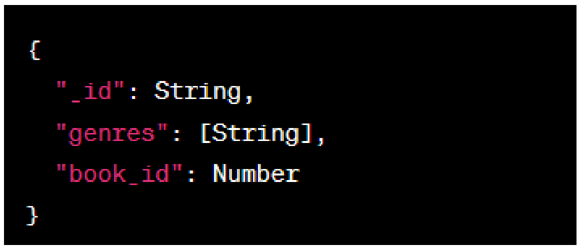
Sample

Before moving on, here are the screenshots of sample document from each collection, and their schema.

Genre:

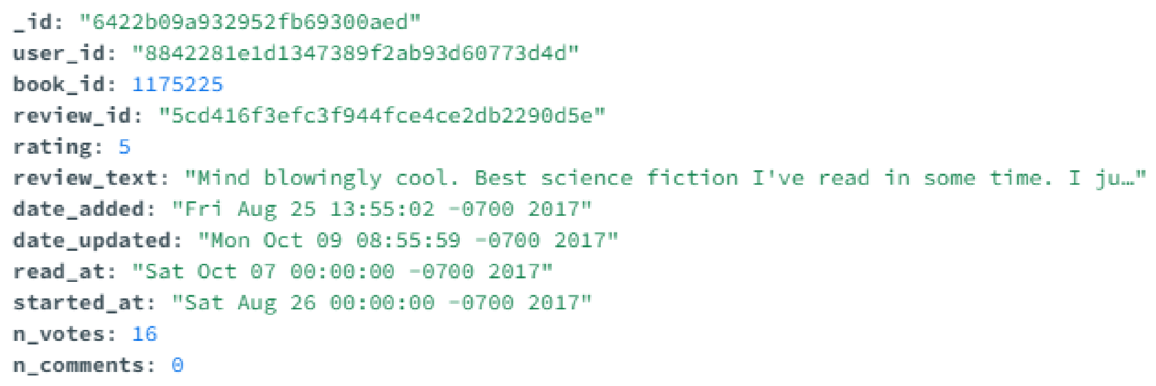


```
_id: "6424452102decca634ec5ee2"
▾ genres: Array
    0: "Array"
book_id: 1
```

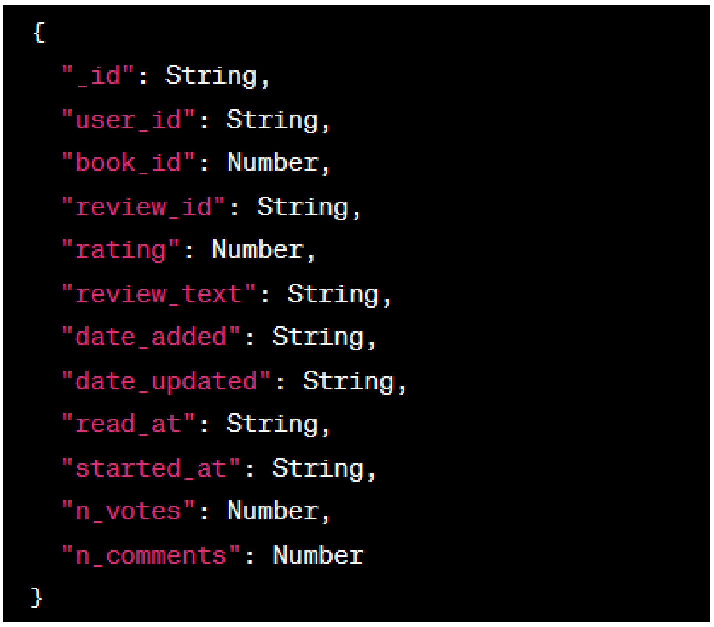


```
{
  "_id": String,
  "genres": [String],
  "book_id": Number
}
```

Review:



```
_id: "6422b09a932952fb69300aed"
user_id: "8842281e1d1347389f2ab93d60773d4d"
book_id: 1175225
review_id: "5cd416f3efc3f944fce4ce2db2290d5e"
rating: 5
review_text: "Mind blowingly cool. Best science fiction I've read in some time. I ju…"
date_added: "Fri Aug 25 13:55:02 -0700 2017"
date_updated: "Mon Oct 09 08:55:59 -0700 2017"
read_at: "Sat Oct 07 00:00:00 -0700 2017"
started_at: "Sat Aug 26 00:00:00 -0700 2017"
n_votes: 16
n_comments: 0
```



```
{
  "_id": String,
  "user_id": String,
  "book_id": Number,
  "review_id": String,
  "rating": Number,
  "review_text": String,
  "date_added": String,
  "date_updated": String,
  "read_at": String,
  "started_at": String,
  "n_votes": Number,
  "n_comments": Number
}
```

Questions:

1. Provide for a minimum of four (4) anomalies. [4 marks each]

    a. Take screenshots of sample documents with the anomaly before it is corrected.

    b. Show the query/queries used to address this anomaly.

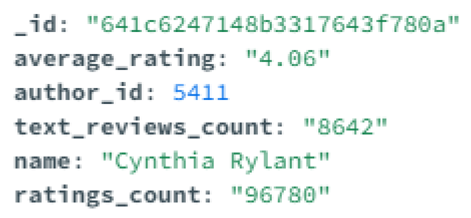
    c. Take screenshots of samples of documents after the anomaly has been corrected.

Solving Part 1:

Solving Anomalies within `author`

Author:

Screenshots of documents before anomalies are corrected



```
_id: "641c6247148b3317643f780a"
average_rating: "4.06"
author_id: 5411
text_reviews_count: "8642"
name: "Cynthia Rylant"
ratings_count: "96780"
```

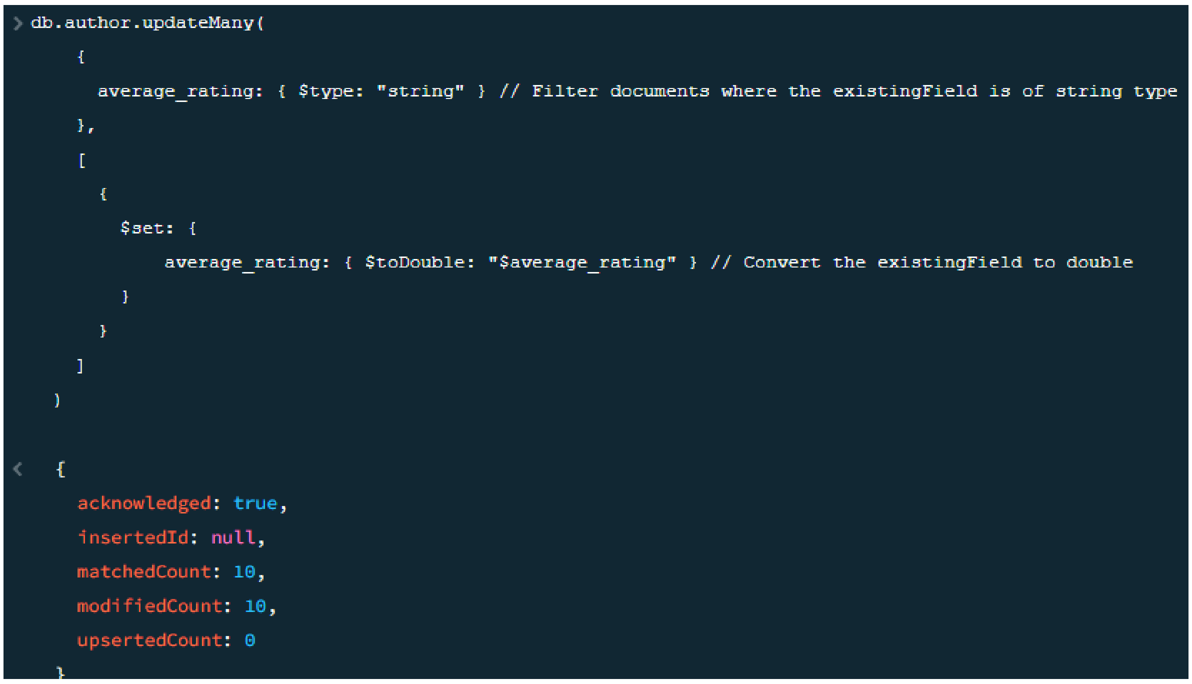
```
{
  "_id": String,
  "average_rating": String,
  "author_id": Number,
  "text_reviews_count": String,
  "name": String,
  "ratings_count": String
}
```

Anomaly#01:

Convert "average_rating" in the Author collection to a numeric data type.

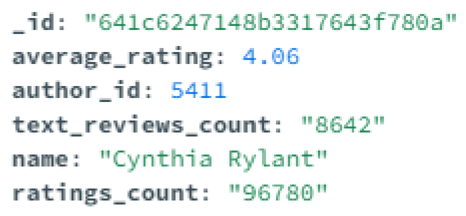
Query:

```
db.author.updateMany(
    {
      average_rating: { $type: "string" } // Filter documents where the
existingField is of string type
    },
    [
      {
        $set: {
            average_rating: { $toDouble: "$average_rating" } // Convert the
existingField to double
        }
      }
    ]
  )
```



```
> db.author.updateMany(
     {
       average_rating: { $type: "string" } // Filter documents where the existingField is of string type
     },
     [
       {
         $set: {
             average_rating: { $toDouble: "$average_rating" } // Convert the existingField to double
         }
       }
     ]
   )

<  {
     acknowledged: true,
     insertedId: null,
     matchedCount: 10,
     modifiedCount: 10,
     upsertedCount: 0
   }
```

Results:



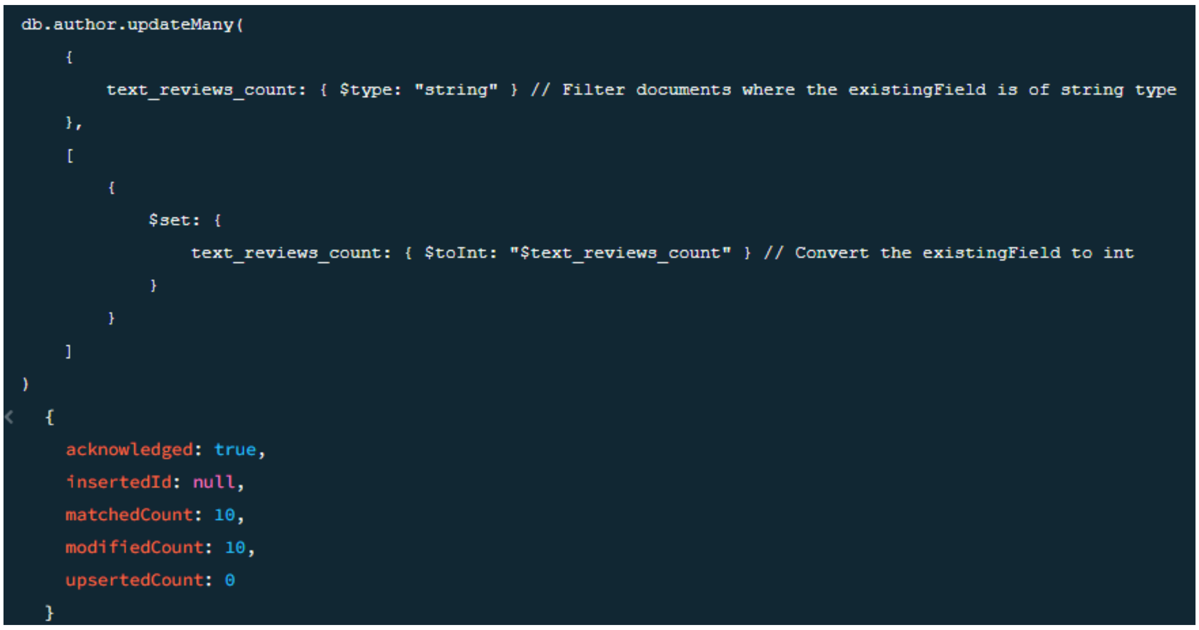
```
_id: "641c6247148b3317643f780a"
average_rating: 4.06
author_id: 5411
text_reviews_count: "8642"
name: "Cynthia Rylant"
ratings_count: "96780"
```

Anomaly#02:

Convert "text_reviews_count" in the Author collection to a numeric data type:

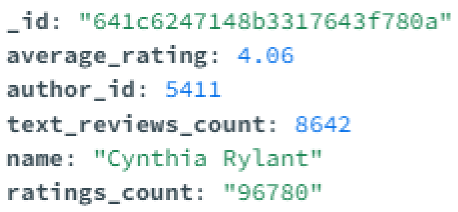
Query:

```
      db.author.updateMany(
    {
        text_reviews_count: { $type: "string" } // Filter documents where the existingField is of string type
    },
    [
        {
            $set: {
                text_reviews_count: { $toDouble: "$text_reviews_count" } // Convert the existingField to int
            }
        }
    ]
)
```



```
db.author.updateMany(
    {
        text_reviews_count: { $type: "string" } // Filter documents where the existingField is of string type
    },
    [
        {
            $set: {
                text_reviews_count: { $toInt: "$text_reviews_count" } // Convert the existingField to int
            }
        }
    ]
)
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 10,
    modifiedCount: 10,
    upsertedCount: 0
  }
```

Results:



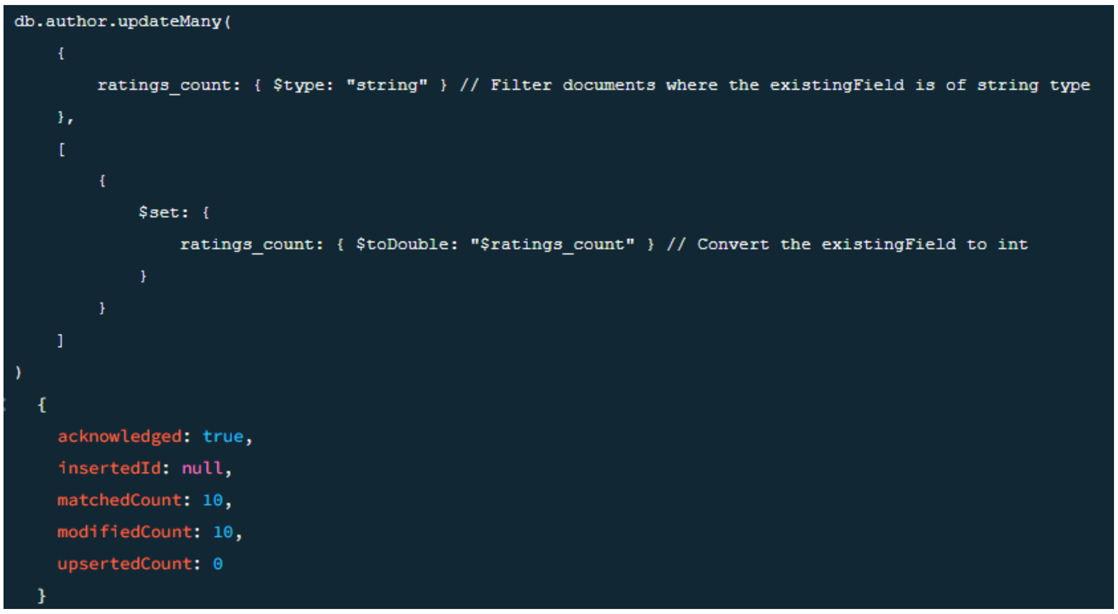
```
_id: "641c6247148b3317643f780a"
average_rating: 4.06
author_id: 5411
text_reviews_count: 8642
name: "Cynthia Rylant"
ratings_count: "96780"
```

Anomaly#03:

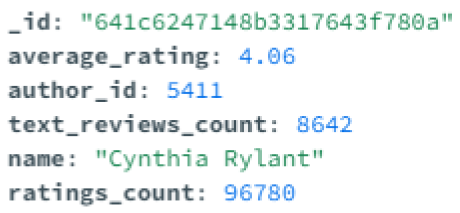
Convert “ratings_count” in the Author collection to a numeric data type:

Query:

```
db.author.updateMany(
    {
        ratings_count: { $type: "string" } // Filter documents where the
existingField is of string type
    },
    [
        {
            $set: {
                ratings_count: { $toDouble: "$ratings_count" } // Convert
the existingField to int
            }
        }
    ]
)
```

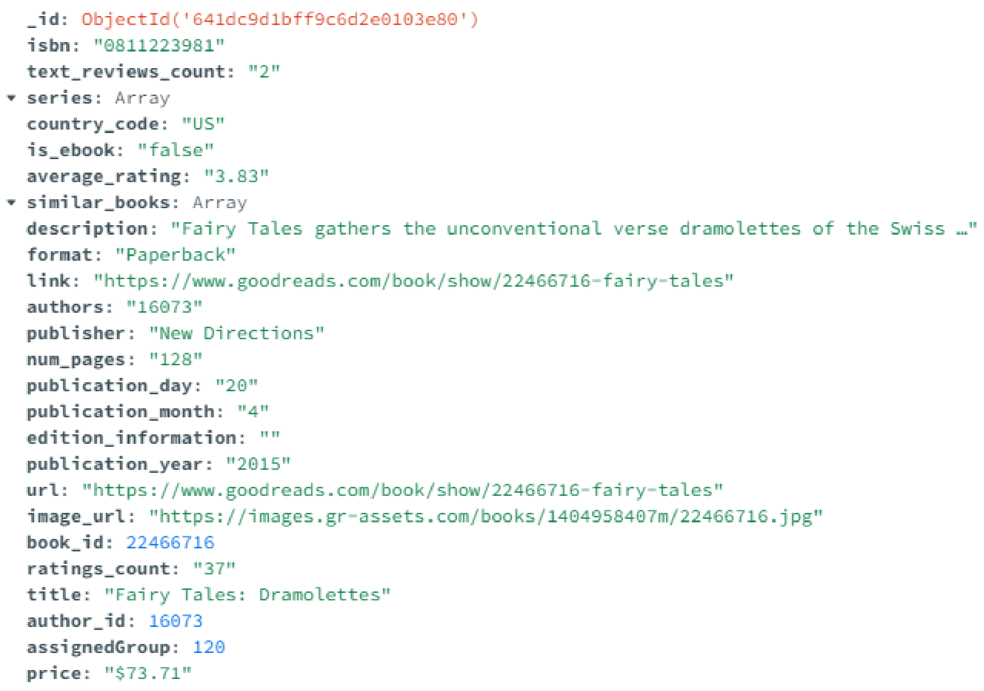

```
db.author.updateMany(
    {
        ratings_count: { $type: "string" } // Filter documents where the existingField is of string type
    },
    [
        {
            $set: {
                ratings_count: { $toDouble: "$ratings_count" } // Convert the existingField to int
            }
        }
    ]
)
  {
    acknowledged: true,
    insertedId: null,
    matchedCount: 10,
    modifiedCount: 10,
    upsertedCount: 0
  }
```

Result:

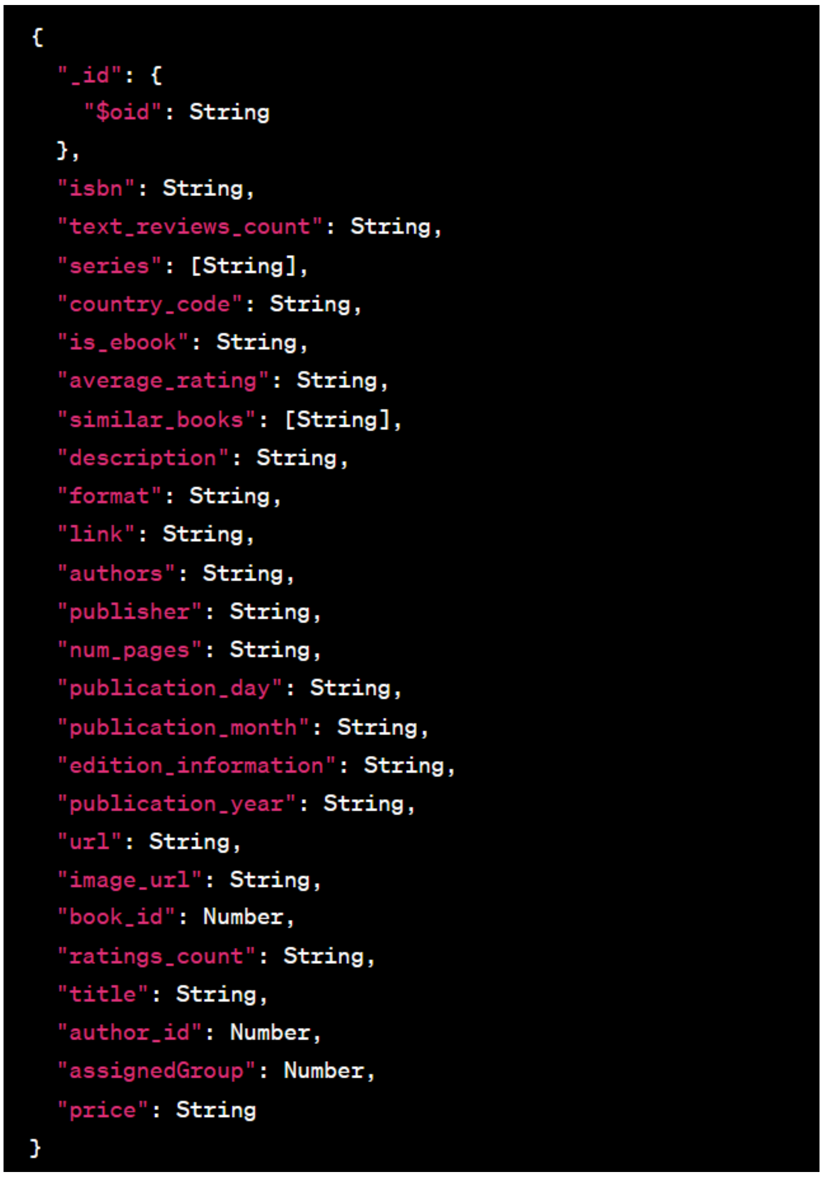

```
_id: "641c6247148b3317643f780a"
average_rating: 4.06
author_id: 5411
text_reviews_count: 8642
name: "Cynthia Rylant"
ratings_count: 96780
```

Solving Book Anomalies:

Book:



```
  _id: ObjectId('641dc9d1bff9c6d2e0103e80')
  isbn: "0811223981"
  text_reviews_count: "2"
▾ series: Array
  country_code: "US"
  is_ebook: "false"
  average_rating: "3.83"
▾ similar_books: Array
  description: "Fairy Tales gathers the unconventional verse dramolettes of the Swiss …"
  format: "Paperback"
  link: "https://www.goodreads.com/book/show/22466716-fairy-tales"
  authors: "16073"
  publisher: "New Directions"
  num_pages: "128"
  publication_day: "20"
  publication_month: "4"
  edition_information: ""
  publication_year: "2015"
  url: "https://www.goodreads.com/book/show/22466716-fairy-tales"
  image_url: "https://images.gr-assets.com/books/1404958407m/22466716.jpg"
  book_id: 22466716
  ratings_count: "37"
  title: "Fairy Tales: Dramolettes"
  author_id: 16073
  assignedGroup: 120
  price: "$73.71"
```



```
{
  "_id": {
    "$oid": String
  },
  "isbn": String,
  "text_reviews_count": String,
  "series": [String],
  "country_code": String,
  "is_ebook": String,
  "average_rating": String,
  "similar_books": [String],
  "description": String,
  "format": String,
  "link": String,
  "authors": String,
  "publisher": String,
  "num_pages": String,
  "publication_day": String,
  "publication_month": String,
  "edition_information": String,
  "publication_year": String,
  "url": String,
  "image_url": String,
  "book_id": Number,
  "ratings_count": String,
  "title": String,
  "author_id": Number,
  "assignedGroup": Number,
  "price": String
}
```

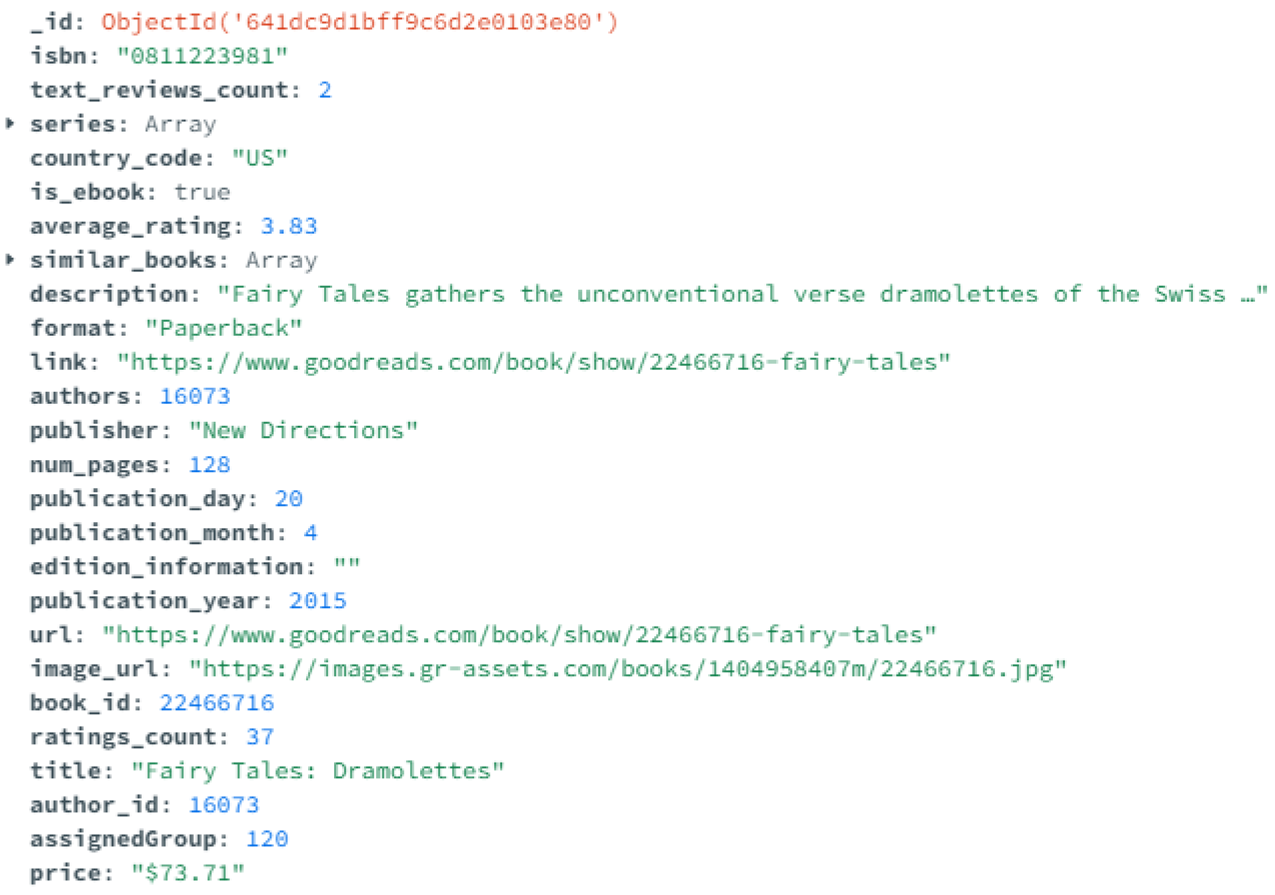
Anomaly#04:

Invalid data types in `book`

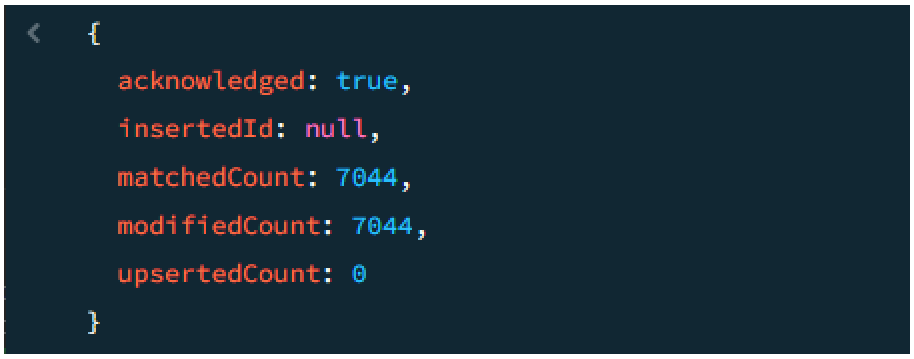
Query:

```
//Adding a condition to the same above query, because there are null / empty values
in fields
db.book.updateMany(
    {
        $or: [
            { authors: { $type: "string" } },
            { text_reviews_count: { $type: "string" } },
            { average_rating: { $type: "string" } },
            { is_ebook: { $type: "string" } },
            { num_pages: { $type: "string" } },
            { publication_day: { $type: "string" } },
            { publication_month: { $type: "string" } },
            { publication_year: { $type: "string" } },
            { ratings_count: { $type: "string" } }
        ]
    },
    [
        {
            $set: {
                text_reviews_count: {
                    $cond: [
                        { $ne: ["$text_reviews_count", ""] },
                        { $toDouble: "$text_reviews_count" },
                        "$text_reviews_count"
                    ]
                },
                authors: {
                    $cond: [
                        { $ne: ["$authors", ""] },
                        { $toInt: "$authors" },
                        "$authors"
                    ]
                },
                average_rating: {
                    $cond: [
                        { $ne: ["$average_rating", ""] },
                        { $toDouble: "$average_rating" },
                        "$average_rating"
                    ]
                },
                is_ebook: {
                    $cond: [
                        { $ne: ["$is_ebook", ""] },
                        { $toBool: "$is_ebook" },
                        "$is_ebook"
                    ]
                },
                num_pages: {
                    $cond: [
                        { $ne: ["$num_pages", ""] },
                        { $toInt: "$num_pages" },
                        "$num_pages"
                    ]
                },
                publication_day: {
                    $cond: [
                        { $ne: ["$publication_day", ""] },
                        { $toInt: "$publication_day" },
                        "$publication_day"
                    ]
                },
                publication_month: {
                    $cond: [
                        { $ne: ["$publication_month", ""] },
                        { $toInt: "$publication_month" },
                        "$publication_month"
                    ]
                },
                publication_year: {
                    $cond: [
                        { $ne: ["$publication_year", ""] },
                        { $toInt: "$publication_year" },
                        "$publication_year"
                    ]
                },
                ratings_count: {
                    $cond: [
                        { $ne: ["$ratings_count", ""] },
                        { $toInt: "$ratings_count" },
                        "$ratings_count"
                    ]
                }
            }
        }
    ]
)
```

Result:



```
  _id: ObjectId('641dc9d1bff9c6d2e0103e80')
  isbn: "0811223981"
  text_reviews_count: 2
▸ series: Array
  country_code: "US"
  is_ebook: true
  average_rating: 3.83
▸ similar_books: Array
  description: "Fairy Tales gathers the unconventional verse dramolettes of the Swiss …"
  format: "Paperback"
  link: "https://www.goodreads.com/book/show/22466716-fairy-tales"
  authors: 16073
  publisher: "New Directions"
  num_pages: 128
  publication_day: 20
  publication_month: 4
  edition_information: ""
  publication_year: 2015
  url: "https://www.goodreads.com/book/show/22466716-fairy-tales"
  image_url: "https://images.gr-assets.com/books/1404958407m/22466716.jpg"
  book_id: 22466716
  ratings_count: 37
  title: "Fairy Tales: Dramolettes"
  author_id: 16073
  assignedGroup: 120
  price: "$73.71"
```



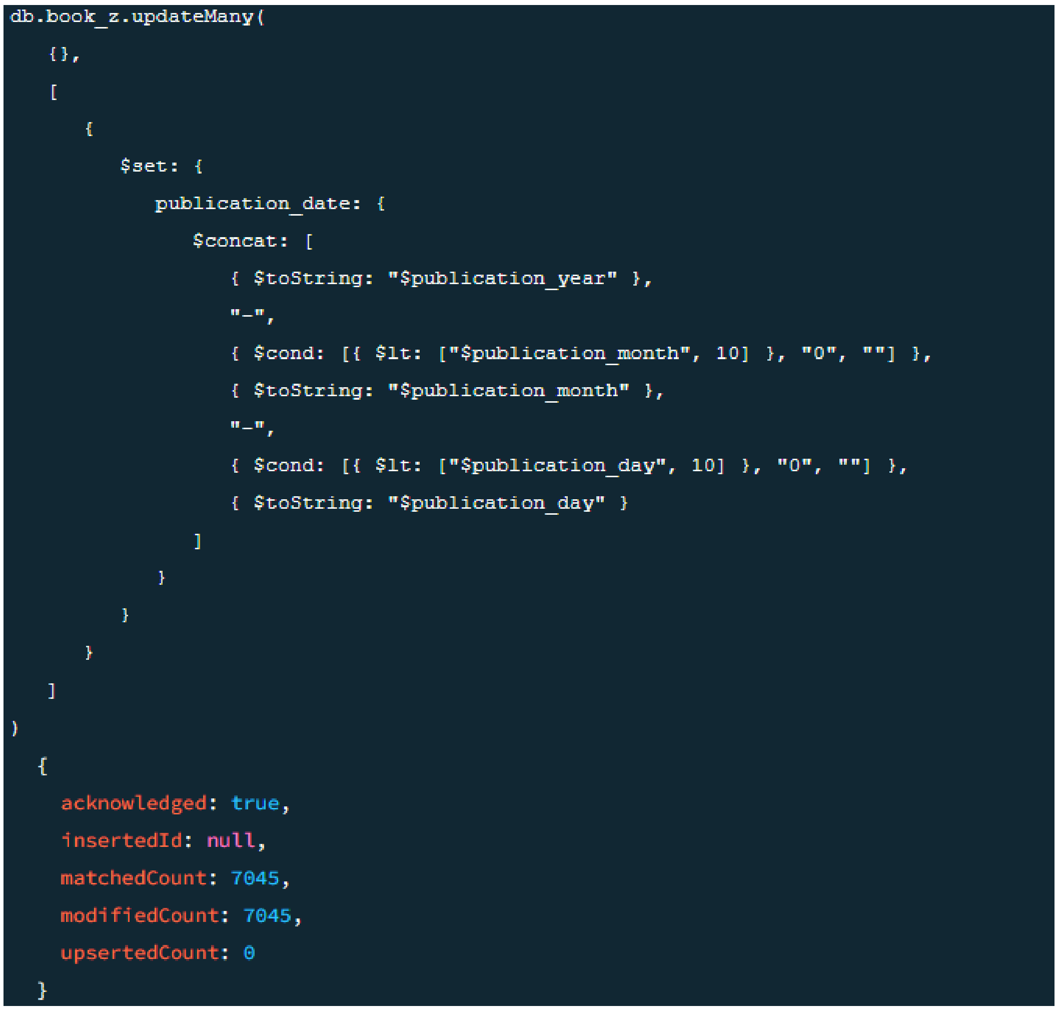
```
<  {
     acknowledged: true,
     insertedId: null,
     matchedCount: 7044,
     modifiedCount: 7044,
     upsertedCount: 0
   }
```

2. Create a new field publication_date in the format YYYY-MM-DD that merges publication_day, publication_month, and publication_year [2 marks]

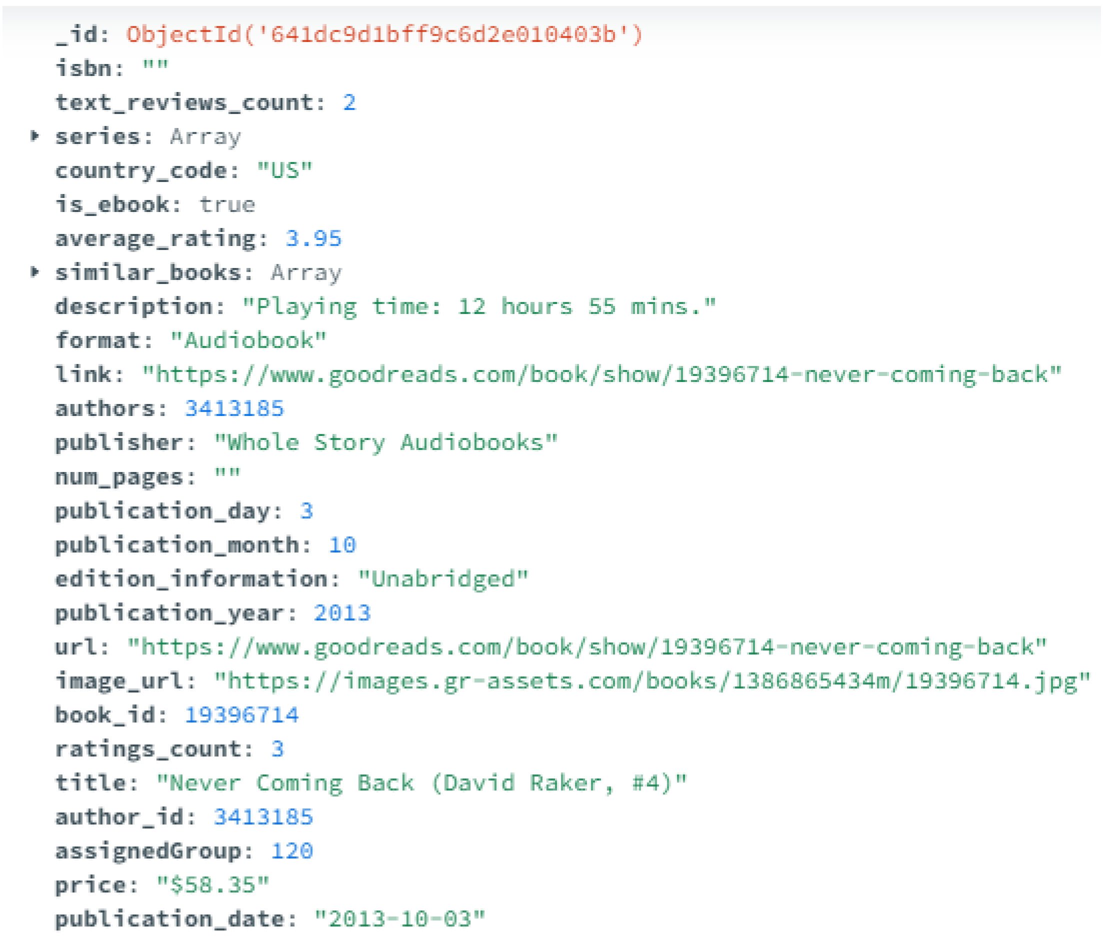
Query:

```
db.book.updateMany(
    {},
    [
      {
        $set: {
          publication_date: {
               $concat: [
                  { $toString: "$publication_year" },
                  "-",
                  { $cond: [{ $lt: ["$publication_month", 10] }, "0", ""] },
                  { $toString: "$publication_month" },
                  "-",
                  { $cond: [{ $lt: ["$publication_day", 10] }, "0", ""] },
                  { $toString: "$publication_day" }
               ]
            }
         }
      }
   ]
)
```

Results:



```
db.book_z.updateMany(
   {},
   [
      {
         $set: {
            publication_date: {
               $concat: [
                  { $toString: "$publication_year" },
                  "-",
                  { $cond: [{ $lt: ["$publication_month", 10] }, "0", ""] },
                  { $toString: "$publication_month" },
                  "-",
                  { $cond: [{ $lt: ["$publication_day", 10] }, "0", ""] },
                  { $toString: "$publication_day" }
               ]
            }
         }
      }
   ]
)
  {
    acknowledged: true,
    insertedId: null,
    matchedCount: 7045,
    modifiedCount: 7045,
    upsertedCount: 0
  }
```



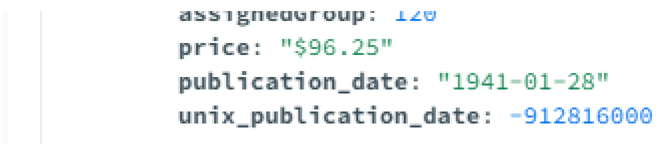
```
  _id: ObjectId('641dc9d1bff9c6d2e010403b')
  isbn: ""
  text_reviews_count: 2
▸ series: Array
  country_code: "US"
  is_ebook: true
  average_rating: 3.95
▸ similar_books: Array
  description: "Playing time: 12 hours 55 mins."
  format: "Audiobook"
  link: "https://www.goodreads.com/book/show/19396714-never-coming-back"
  authors: 3413185
  publisher: "Whole Story Audiobooks"
  num_pages: ""
  publication_day: 3
  publication_month: 10
  edition_information: "Unabridged"
  publication_year: 2013
  url: "https://www.goodreads.com/book/show/19396714-never-coming-back"
  image_url: "https://images.gr-assets.com/books/1386865434m/19396714.jpg"
  book_id: 19396714
  ratings_count: 3
  title: "Never Coming Back (David Raker, #4)"
  author_id: 3413185
  assignedGroup: 120
  price: "$58.35"
  publication_date: "2013-10-03"
```

3. Create another field in the books collection **unix_publication_date** that converts **publication_date** to Unix format. [2 marks]

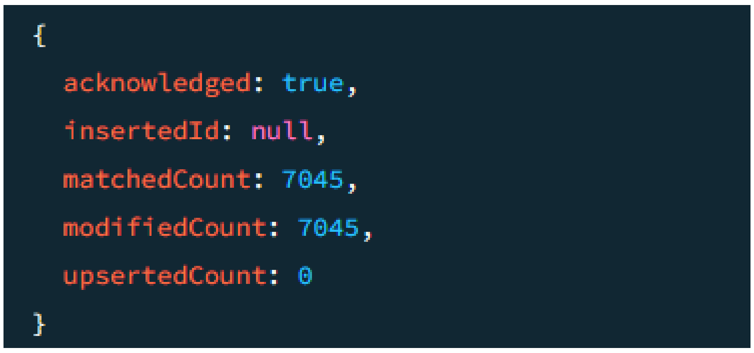
Query:

```
 db.book_z.aggregate([
    {
      $addFields: {
        unix_publication_date: {
          $cond: [
            {
              $and: [
                { $ne: ["$publication_date", null] },
                { $ne: ["$publication_date", ""] },
                { $regexMatch: { input: "$publication_date", regex: /^\d{4}-\d{2}-
\d{2}$/ } }
              ]
            },
            {
              $divide: [
                { $subtract: [{ $toDate: { $concat: ["$publication_date",
"T00:00:00Z"] } }, new Date("1970-01-01T00:00:00Z")] },
                1000
              ]
            },
            null
          ]
        }
      }
    },
    {
      $out: "book_z" // Save the updated documents back to the collection
    }
  ])
```

Result:



```
            assignedGroup: 120
            price: "$96.25"
            publication_date: "1941-01-28"
            unix_publication_date: -912816000
```



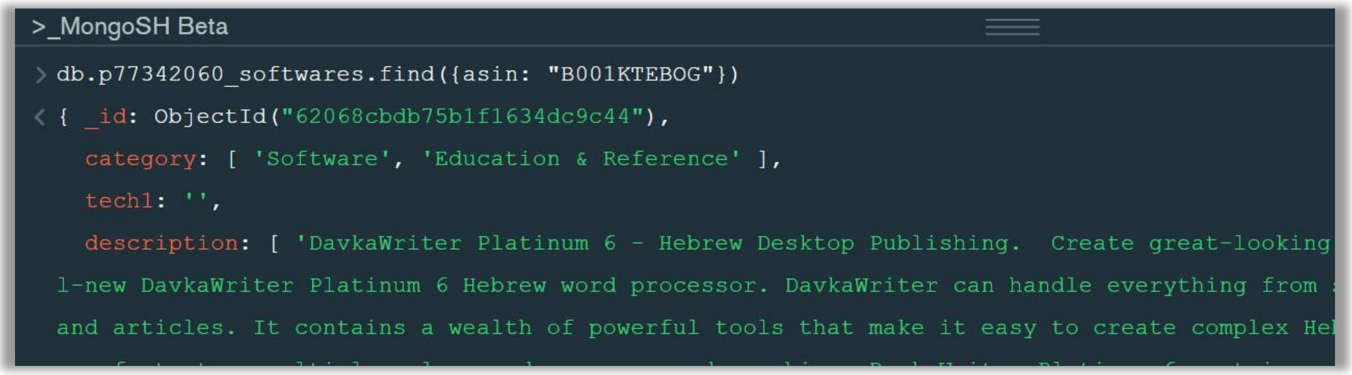
```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 7045,
  modifiedCount: 7045,
  upsertedCount: 0
}
```

5.0 QUERYING THE COLLECTIONS [20 marks]

Write the following queries (Q1 to Q4) against the collections that you personalised and loaded you're your local MongoDB. For each query, submit the MongoDB command in plain text, AND present it with its results as a screenshot showing the command and the documents returned. **For example:**

//Q1001. Find the details of the product identified as " **B001KTEBOG** ".

db.p77342060_softwares.find({asin:"B001KTEBOG"})



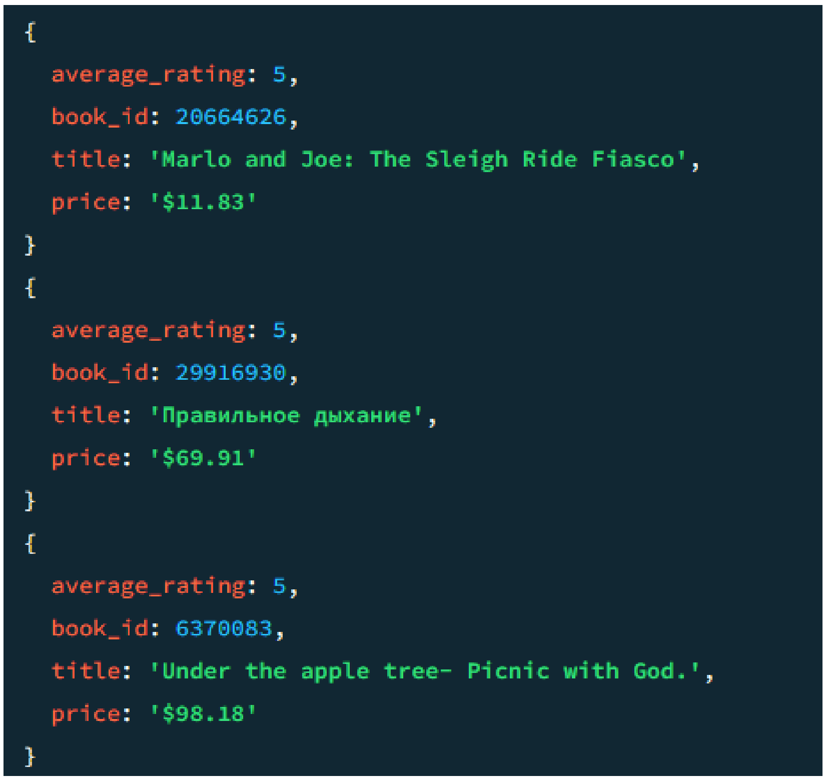
```
>_MongoSH Beta
> db.p77342060_softwares.find({asin: "B001KTEBOG"})
< { _id: ObjectId("62068cbdb75b1f1634dc9c44"),
    category: [ 'Software', 'Education & Reference' ],
    tech1: '',
    description: [ 'DavkaWriter Platinum 6 - Hebrew Desktop Publishing.  Create great-looking
  l-new DavkaWriter Platinum 6 Hebrew word processor. DavkaWriter can handle everything from
  and articles. It contains a wealth of powerful tools that make it easy to create complex He
```

If a lot of documents are returned, show the first set of documents that are displayed. Ensure that what is displayed answers the requirements and demonstrates that the query is correct. You must do this to gain all the marks available for a question.

Q1. Find top 3 books with highest average rating above 3.5 and are not e-books. Display only book id, title, price, and average rating. [5 marks]

Query:

```
db.book_z.find(
    { average_rating: { $gt: 3.5, $ne: null, $ne: "" }, is_ebook: false},
    {
        book_id: 1,
        title: 1,
        price: 1,
        average_rating: 1,
        _id: 0
    }
).sort({ average_rating: -1 }).limit(3)
```



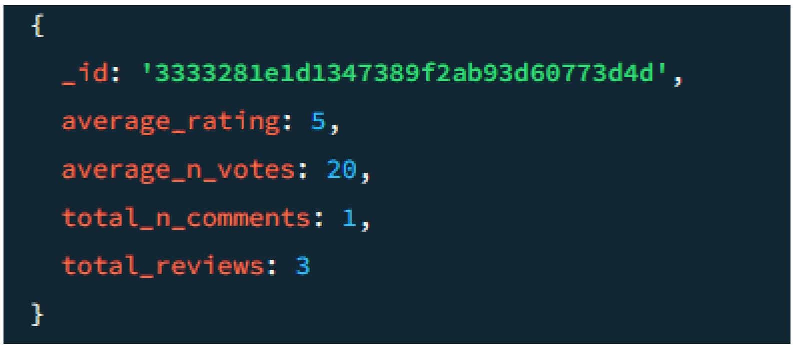
```
{
  average_rating: 5,
  book_id: 20664626,
  title: 'Marlo and Joe: The Sleigh Ride Fiasco',
  price: '$11.83'
}
{
  average_rating: 5,
  book_id: 29916930,
  title: 'Правильное дыхание',
  price: '$69.91'
}
{
  average_rating: 5,
  book_id: 6370083,
  title: 'Under the apple tree- Picnic with God.',
  price: '$98.18'
}
```

Q2. Who is the most famous reviewer in your dataset? Most famous reviewer have the highest number of 5-rated reviews in your datasets. Display the **user_id, average rating, average n_votes, total n_comments and total number of reviews** written by this reviewer. [6 marks]

Query:

```
db.review.aggregate([
    {
      $match: { rating: 5 } // Filter only 5-rated reviews
    },
    {
      $group: {
        _id: "$user_id", // Group by user_id
        average_rating: { $avg: "$rating" }, // Calculate average rating
        average_n_votes: { $avg: "$n_votes" }, // Calculate average n_votes
        total_n_comments: { $sum: "$n_comments" }, // Calculate total n_comments
        total_reviews: { $sum: 1 } // Calculate total number of reviews
      }
    },
    {
      $sort: { total_reviews: -1 } // Sort by total_reviews in descending order
    },
    {
      $limit: 1 // Retrieve only the top result
    },
    {
      $project: {
        _id: 1, // Include user_id
        average_rating: 1, // Include average rating
        average_n_votes: 1, // Include average n_votes
        total_n_comments: 1, // Include total n_comments
        total_reviews: 1 // Include total reviews
      }
    }
  ])
```

Result:

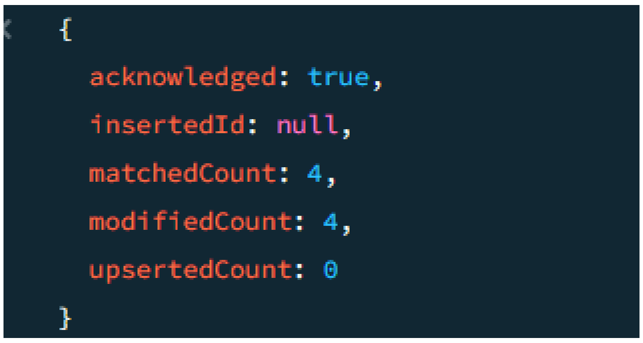


```
{
  _id: '3333281e1d1347389f2ab93d60773d4d',
  average_rating: 5,
  average_n_votes: 20,
  total_n_comments: 1,
  total_reviews: 3
}
```

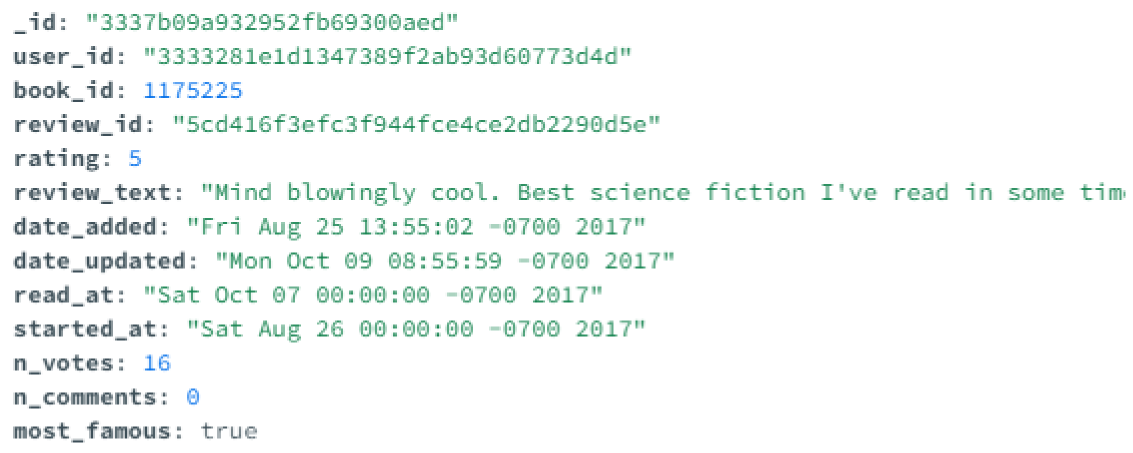
Q3. Update all reviews written by the most famous reviewer from Q2 by adding a new field named **most_famous** and set its value to **true.** [4 mark]

Query:

```
var mostFamousReviewer = db.review.aggregate([
  {
    $group: {
      _id: "$user_id",
      average_rating: { $avg: "$rating" },
      average_n_votes: { $avg: "$n_votes" },
      total_n_comments: { $sum: "$n_comments" },
      total_reviews: { $sum: 1 }
    }
  },
  { $sort: { total_reviews: -1 } },
  { $limit: 1 }
]).next();

// Step 2: Update reviews written by the most famous reviewer
db.review.updateMany(
  { user_id: mostFamousReviewer._id },
  { $set: { most_famous: true } }
);
```



```
  {
    acknowledged: true,
    insertedId: null,
    matchedCount: 4,
    modifiedCount: 4,
    upsertedCount: 0
  }
```



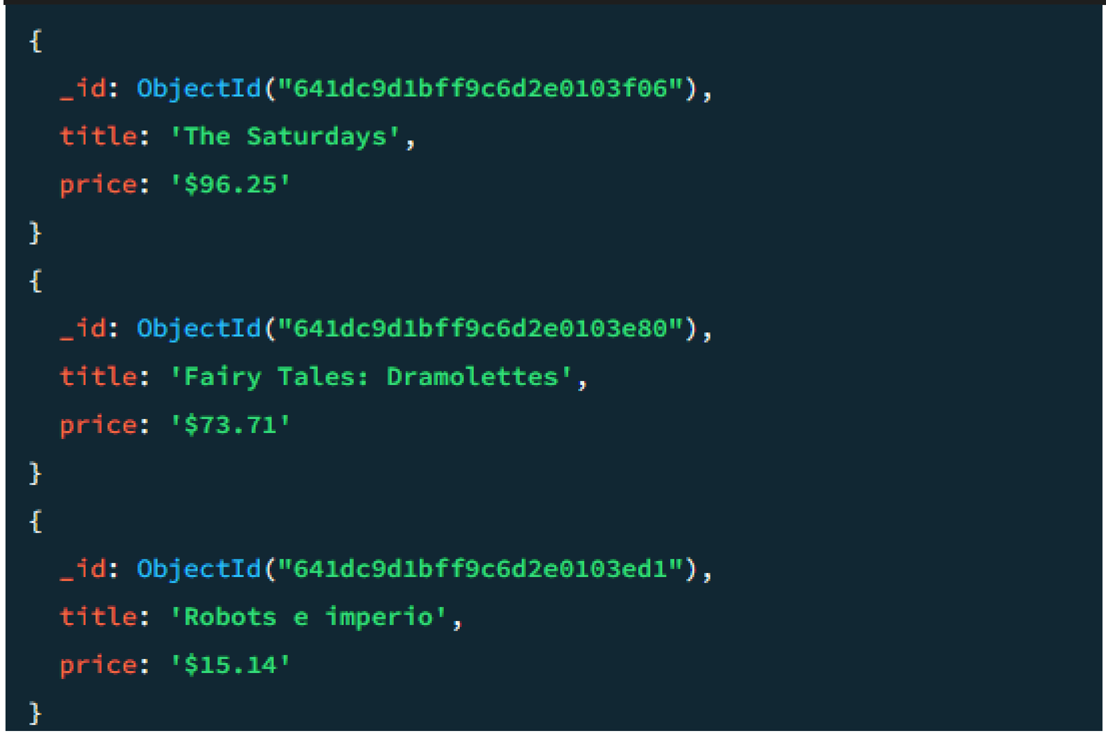
```
_id: "3337b09a932952fb69300aed"
user_id: "3333281e1d1347389f2ab93d60773d4d"
book_id: 1175225
review_id: "5cd416f3efc3f944fce4ce2db2290d5e"
rating: 5
review_text: "Mind blowingly cool. Best science fiction I've read in some tim
date_added: "Fri Aug 25 13:55:02 -0700 2017"
date_updated: "Mon Oct 09 08:55:59 -0700 2017"
read_at: "Sat Oct 07 00:00:00 -0700 2017"
started_at: "Sat Aug 26 00:00:00 -0700 2017"
n_votes: 16
n_comments: 0
most_famous: true
```

Q4. Using reviews and books collections, find the title and price of 3 most expensive books reviewed by the most famous reviewer. No other product details are required. [5 marks]

Query:

```
var mostFamousReviewer = db.review.findOne({ most_famous: true });
```

```
if (mostFamousReviewer) {
  var mostFamousReviewerId = mostFamousReviewer.user_id;
  const reviewIds = [];
  db.review.find({ user_id: mostFamousReviewerId }).forEach(review => {
    reviewIds.push(review.book_id);
  });
  db.book.find({ book_id: { $in: reviewIds } }, { title: 1, price: 1 }).sort({ price: -1
}).limit(3)
}
```



```
{
  _id: ObjectId("641dc9d1bff9c6d2e0103f06"),
  title: 'The Saturdays',
  price: '$96.25'
}
{
  _id: ObjectId("641dc9d1bff9c6d2e0103e80"),
  title: 'Fairy Tales: Dramolettes',
  price: '$73.71'
}
{
  _id: ObjectId("641dc9d1bff9c6d2e0103ed1"),
  title: 'Robots e imperio',
  price: '$15.14'
}
```

6.0 IMPLEMENT AND EXPLAIN INDEX FOR THE DATABASE [15 marks]

1. Identify the chosen query from Section 5 and explain/justify your choice of index. Why do you think indexing could improve the query. [5 marks]

   Answer:

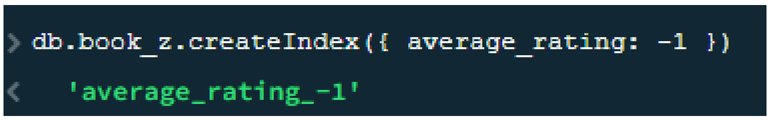
   Based on the queries that I provided in Section-05, there is no explicit mention of indexing.

   Indexing can greatly improve the performance of queries by creating efficient data structures that allow for faster data retrieval.

   To optimize the performance of the queries, I can consider creating indexes on the fields that are frequently used for filtering or sorting. In this case, some potential fields that could benefit from indexing are **average_rating** and **is_ebook** in the **book** collection, and **rating** and **user_id** in the **review** collection.

Creating indexes on these fields can significantly improve the execution time of the queries by enabling the database to quickly locate and retrieve the relevant data. However, the decision to create indexes should be based on the specific requirements and usage patterns of your application.

2. Implement one index that would improve the querying of the database based on one of the queries (Q1-Q4). [2 marks]



```
> db.book_z.createIndex({ average_rating: -1 })
<   'average_rating_-1'
```

3. Present the execution plan of the selected query before the index and after the index has been created. Compare and discuss the execution plans to support your choice and summarise your findings.

Query Code:

```
db.book_z.find(
  { average_rating: { $gt: 3.5, $ne: null, $ne: "" }, is_ebook: false },
  {
    book_id: 1,
    title: 1,
    price: 1,
    average_rating: 1,
    _id: 0
  }
).sort({ average_rating: -1 }).limit(3).explain()
```

[8 marks]

Response Code:

```
{
    explainVersion: '1',
    queryPlanner: {
      namespace: 'goodReads.book_z',
      indexFilterSet: false,
      parsedQuery: {
        '$and': [
          {
            is_ebook: {
              '$eq': false
            }
          },
          {
            average_rating: {
```

```
            '$gt': 3.5
          }
        },
        {
          average_rating: {
            '$not': {
              '$eq': ''
            }
          }
        }
      ]
    },
    queryHash: '5D130F30',
    planCacheKey: 'E3512104',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'PROJECTION_SIMPLE',
      transformBy: {
        book_id: 1,
        title: 1,
        price: 1,
        average_rating: 1,
        _id: 0
      },
      inputStage: {
        stage: 'SORT',
        sortPattern: {
          average_rating: -1
        },
        memLimit: 104857600,
        limitAmount: 3,
        type: 'simple',
        inputStage: {
          stage: 'COLLSCAN',
          filter: {
            '$and': [
              {
                is_ebook: {
                  '$eq': false
                }
              },
              {
                average_rating: {
                  '$gt': 3.5
                }
              },
              {
                average_rating: {
                  '$not': {
                    '$eq': ''
                  }
                }
              }
            ]
          },
          direction: 'forward'
        }
      }
    },
    rejectedPlans: []
  },
  command: {
    find: 'book_z',
    filter: {
      average_rating: {
        '$gt': 3.5,
        '$ne': ''
      },
      is_ebook: false
    },
    sort: {
      average_rating: -1
    },
    projection: {
      book_id: 1,
      title: 1,
      price: 1,
      average_rating: 1,
      _id: 0
    },
    limit: 3,
    '$db': 'goodReads'
  },
  serverInfo: {
    host: 'DESKTOP-D3SC6RH',
    port: 27017,
    version: '6.0.5',
    gitVersion: 'c9a99c120371d4d4c52cbb15dac34a36ce8d3b1d'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
    },
    ok: 1
  }



//After indexing:



{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'goodReads.book_z',
    indexFilterSet: false,
    parsedQuery: {
      '$and': [
        {
          is_ebook: {
            '$eq': false
          }
        },
        {
          average_rating: {
            '$gt': 3.5
          }
        },
        {
          average_rating: {
            '$not': {
              '$eq': ''
            }
          }
        }
      ]
    },
    queryHash: '5D130F30',
    planCacheKey: '4CDB477F',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'LIMIT',
      limitAmount: 3,
      inputStage: {
        stage: 'PROJECTION_SIMPLE',
        transformBy: {
          book_id: 1,
          title: 1,
          price: 1,
          average_rating: 1,
          _id: 0
        },
        inputStage: {
          stage: 'FETCH',
          filter: {
            is_ebook: {
              '$eq': false
            }
          },
          inputStage: {
            stage: 'IXSCAN',
            keyPattern: {
              average_rating: -1
            },
            indexName: 'average_rating_-1',
            isMultiKey: false,
            multiKeyPaths: {
              average_rating: []
            },
            isUnique: false,
            isSparse: false,
            isPartial: false,
            indexVersion: 2,
            direction: 'forward',
            indexBounds: {
              average_rating: [
                '[inf.0, 3.5)'
              ]
            }
          }
        }
      }
    },
    rejectedPlans: []
  },
  command: {
    find: 'book_z',
    filter: {
      average_rating: {
        '$gt': 3.5,
        '$ne': ''
      },
      is_ebook: false
    },
    sort: {
      average_rating: -1
    },
    projection: {
      book_id: 1,
      title: 1,
      price: 1,
      average_rating: 1,
      _id: 0
    },
    limit: 3,
    '$db': 'goodReads'
  },
  serverInfo: {
    host: 'DESKTOP-D3SC6RH',
    port: 27017,
    version: '6.0.5',
    gitVersion: 'c9a99c120371d4d4c52cbb15dac34a36ce8d3b1d'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
  },
  ok: 1
}
```

Explanation and findings:

Before Indexing:

Based on the provided execution plan before indexing, we can observe the following:

1. Stage: The winning plan indicates that the execution starts with a COLLSCAN (collection scan) stage, which means a full collection scan is performed without utilizing an index.

2. Sorting: After the collection scan, a SORT stage is applied to sort the documents based on the average_rating field in descending order. This sorting operation requires additional memory resources.

3. Limit: Finally, a LIMIT stage is applied to restrict the result to the top 3 documents.

4. Rejected Plans: No rejected plans are mentioned in the execution plan.

Considering the execution plan, we can see that without an index, the query has to perform a full collection scan and apply sorting on the entire dataset. This approach can be inefficient and time-consuming, especially for larger collections.

To optimize the query execution, we can create an index on the average_rating field. This index will enable the query to efficiently filter and sort the documents based on the average_rating criteria.

After Indexing:

After indexing the **average_rating** field in the **book_z** collection, the new execution plan shows the following changes:

1. Stage: The winning plan now starts with an **IXSCAN** (index scan) stage, indicating that the index on the **average_rating** field is utilized for filtering and sorting.
2. Index Usage: The **IXSCAN** stage specifies the index name (**average_rating_-1**) and the index bounds (**[inf.0, 3.5)**), which indicates that the index is used to efficiently filter documents with **average_rating** greater than 3.5.
3. Fetch Stage: After the index scan, a **FETCH** stage is applied to retrieve the matching documents from the collection based on the filter **{ is_ebook: { $eq: false } }**.
4. Projection and Limit: Finally, a **PROJECTION_SIMPLE** stage is applied to project and limit the desired fields (**book_id**, **title**, **price**, **average_rating**) and the result is limited to 3 documents using the **LIMIT** stage.

By utilizing the index, the query execution plan has improved significantly. The index allows for efficient filtering based on the **average_rating** field, reducing the number of documents that need to be examined. As a result, the query should execute faster and consume fewer resources compared to the previous execution plan without an index.

Overall, the index on the **average_rating** field enhances the query performance by leveraging index scans and minimizing the amount of data that needs to be processed.

Differences and basic findings:

Comparing the responses before and after indexing, we can observe the following differences:

Before Indexing:

- The winning plan starts with a **COLLSCAN** (collection scan) stage, which means that it scans the entire collection without utilizing any indexes.
- The documents are filtered using the **$and** operator and the conditions on **is_ebook** and **average_rating**.
- The filtered documents are then sorted by **average_rating** in descending order using a **SORT** stage.
- Finally, the sorted documents are limited to 3 using a **LIMIT** stage.

After Indexing:

- The winning plan starts with an **IXSCAN** (index scan) stage, indicating that it utilizes the index on the **average_rating** field.
- The index scan filters the documents based on the condition **$gt: 3.5** for **average_rating**.

- The filtered documents are fetched using a **FETCH** stage based on the additional filter **is_ebook: { $eq: false }**.
- The fetched documents are then projected and limited to 3 using a **PROJECTION_SIMPLE** stage and a **LIMIT** stage, respectively.

Key Differences:

- Before indexing, the query performed a collection scan, which is less efficient and requires scanning all documents in the collection. After indexing, the query utilizes an index scan, which significantly improves performance by directly accessing the relevant documents based on the indexed field.
- The use of indexing allows for more selective and efficient filtering of documents, reducing the number of documents that need to be processed.
- The query plan after indexing is more optimized, as it starts with an index scan followed by a fetch and projection, resulting in faster execution and reduced resource consumption.

In summary, indexing the **average_rating** field improves the query performance by enabling index scans and selective filtering. This leads to a more efficient execution plan, reducing the query execution time and resource utilization.

Comparision of execution Plan:

Before Indexing:

```
//Execution plan bfore indexing
{
  ...
  "winningPlan": {
    "stage": "PROJECTION_SIMPLE",
    "transformBy": {
      "book_id": 1,
      "title": 1,
      "price": 1,
      "average_rating": 1,
      "_id": 0
    },
    "inputStage": {
      "stage": "SORT",
      "sortPattern": {
        "average_rating": -1
      },
      ...
      "inputStage": {
        "stage": "COLLSCAN",
        ...
      }
    }
  },
  ...
}
```

After Indexing:

```
//After indexing

{
  ...
  "winningPlan": {
    "stage": "LIMIT",
    "limitAmount": 3,
    "inputStage": {
      "stage": "PROJECTION_SIMPLE",
      "transformBy": {
        "book_id": 1,
        "title": 1,
        "price": 1,
        "average_rating": 1,
        "_id": 0
      },
      "inputStage": {
        "stage": "FETCH",
        "filter": {
          "is_ebook": {
            "$eq": false
          }
        },
        "inputStage": {
          "stage": "IXSCAN",
          "keyPattern": {
            "average_rating": -1
          },
          ...
        }
      }
    }
  },
  ...
}
```

Discussion:

Comparison and Discussion: Before Indexing:

- The winning plan starts with a SORT stage to sort the documents based on average_rating in descending order.
- The sorting operation requires scanning all documents using a COLLSCAN (collection scan) stage, which can be inefficient for large collections.

After Indexing:

- The winning plan starts with an IXSCAN (index scan) stage, utilizing the index on the average_rating field.
- The index scan retrieves the documents based on the condition $gt: 3.5 for average_rating.
- The fetched documents are further filtered using the condition is_ebook: { $eq: false } in the FETCH stage.
- Finally, the filtered documents are projected and limited to 3 using a PROJECTION_SIMPLE stage and a LIMIT stage, respectively.

Findings:

- After creating an index on the average_rating field, the execution plan is more optimized.
- The index scan allows for efficient retrieval of documents based on the indexed field, avoiding the need for a collection scan.
- The use of indexing significantly improves the performance of the query by reducing the number of documents that need to be processed.
- The query execution time and resource utilization are expected to be lower with the indexed version of the query.

In summary, creating an index on the average_rating field improves the query performance by enabling index scans and more efficient filtering. This results in faster execution and reduced resource consumption compared to the query without an index.

7.0 RE-DESIGN THE DATABASE USING AGGREGATE DATA MODELLING [20 marks]

Write code in MongoDB to automatically embed the details of authors from authors collection and genres from genres collection with their corresponding book in the books collection. This is an aggregate data modelling (ADM) question.

Therefore, the books collection will contain the book data, authors data and genre data in a single collection of books. This requires the following tasks:

1. Make new copy of the pxxxxxx_books collection and name it pxxxxxx_books_adm. [1 mark]

   Query:

```
//Part 01
```

```
db.book.aggregate([
    { $match: {} }, // Match all documents in the book collection
    { $out: "books_adm" } // Output the results to the books_adm collection
  ])
```

2. Embed all the authors and genres of books into their corresponding book using the new pxxxxxx_books_adm collection. [6 marks]

Query:

```
  //Final Part 02:
  //Query result stored in a separate collection without modifyning the original book collection
  db.book.aggregate([
    {
      $lookup: {
        from: "author",
        localField: "author_id",
        foreignField: "author_id",
        as: "author"
      }
    },
    {
      $unwind: "$author"
    },
    {
      $lookup: {
        from: "genre",
        localField: "book_id",
        foreignField: "book_id",
        as: "genre"
      }
    },
    {
      $project: {
        _id: 0,
        isbn: 1,
        text_reviews_count: 1,
        series: 1,
        country_code: 1,
        is_ebook: 1,
        average_rating: 1,
        similar_books: 1,
        description: 1,
        format: 1,
        link: 1,
        authors: 1,
        publisher: 1,
        num_pages: 1,
        publication_day: 1,
        publication_month: 1,
        edition_information: 1,
        publication_year: 1,
        url: 1,
        image_url: 1,
        book_id: 1,
        ratings_count: 1,
        title: 1,
        assignedGroup: 1,
        price: 1,
        author: "$author",
        genre: {
          $ifNull: [{ $arrayElemAt: ["$genre.genres", 0] }, []]
        }
      }
    },
    {
      $out: "new_book_adm"
    }
  ]);



  //null value check
  // Iterate over the documents in the new_book_adm collection
    db.new_book_adm.find().forEach(function(newDoc) {
    // Find the corresponding document in the books_adm collection based on
author_id
    var query = { "author_id": newDoc.author.author_id };
    var existingDoc = db.books_adm.findOne(query);

    // Preserve the original _id field in the new document if it exists
    if (existingDoc && existingDoc._id) {
      newDoc._id = existingDoc._id;
    } else {
      delete newDoc._id; // Remove _id field if it's null or empty
    }

    // Replace the document in the books_adm collection with the one from
new_book_adm
    if (existingDoc) {
      db.books_adm.replaceOne(query, newDoc);
    }
  });

    db.new_book_adm.drop()
```

The query performs an aggregation operation on the **book** collection. It first performs a **$lookup** stage to join the **author** collection based on the **author_id** field, and then unwinds the resulting array using **$unwind**. Next, it performs another **$lookup** stage to join the **genre** collection based on the **book_id** field.

The **$project** stage is used to shape the output documents, including selecting and renaming specific fields. It also handles null values using **$ifNull** to ensure that the **genre** field contains an array.

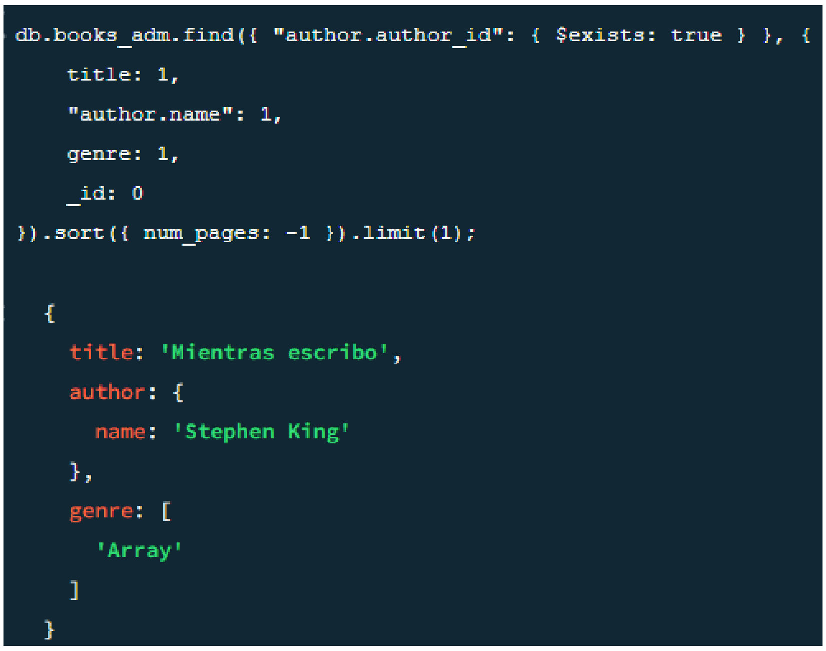
Finally, the **$out** stage is used to store the resulting documents in a new collection named **new_book_adm**.

After creating the **new_book_adm** collection, the script iterates over the documents in this collection. For each document, it searches for the corresponding document in the **books_adm** collection based on the **author_id**. If a match is found, the original **_id** field is preserved in the new document. If the **_id** field is null or empty, it is removed. Then, the document in the **books_adm** collection is replaced with the updated document from **new_book_adm**.

Finally, the **new_book_adm** collection is dropped to clean up the temporary collection.

3. To verify that the changes to the new products collection were successful, display the title, authors, and genre of the book with highest number of pages. [1 mark]

   Result:



```
db.books_adm.find({ "author.author_id": { $exists: true } }, {
    title: 1,
    "author.name": 1,
    genre: 1,
    _id: 0
}).sort({ num_pages: -1 }).limit(1);

  {
    title: 'Mientras escribo',
    author: {
      name: 'Stephen King'
    },
    genre: [
      'Array'
    ]
  }
```

   Query:

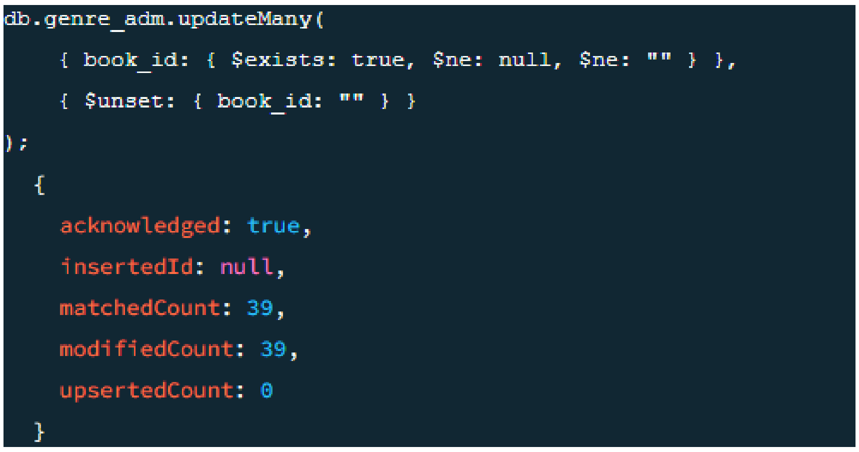
```
db.books_adm.find({ "author.author_id": { $exists: true } }, {
          title: 1,
    "author.name": 1,
    genre: 1,
    _id: 0
}).sort({ num_pages: -1 }).limit(1);
```

4. Note that there is no longer need to reference the book_id inside each genre or author_id inside each book now that they are part of the book in pxxxxxx_books_adm collection. Show that these are removed. [2 marks]

   Query:

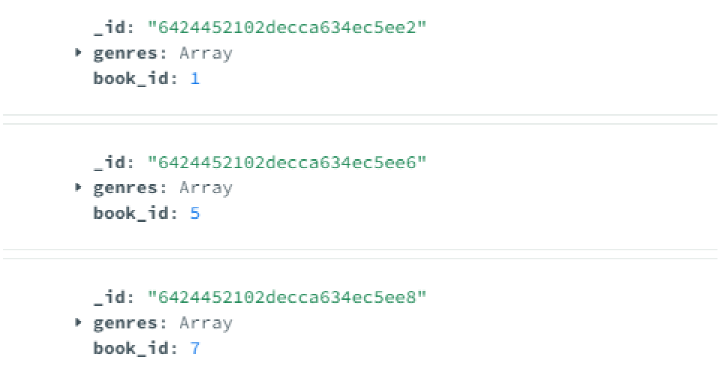
```
db.genre.updateMany(
    { book_id: { $exists: true, $ne: null, $ne: "" } },
    { $unset: { book_id: "" } }
);

db.book.updateMany(
    { author_id: { $exists: true, $ne: null, $ne: "" } },
    { $unset: { author_id: "" } }
);
```

Query Result: Genre



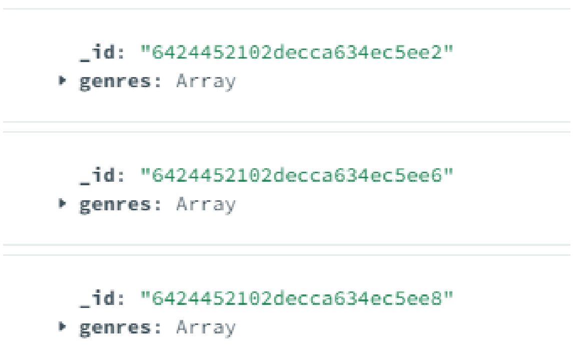
```
db.genre_adm.updateMany(
    { book_id: { $exists: true, $ne: null, $ne: "" } },
    { $unset: { book_id: "" } }
);
  {
    acknowledged: true,
    insertedId: null,
    matchedCount: 39,
    modifiedCount: 39,
    upsertedCount: 0
  }
```

Before:



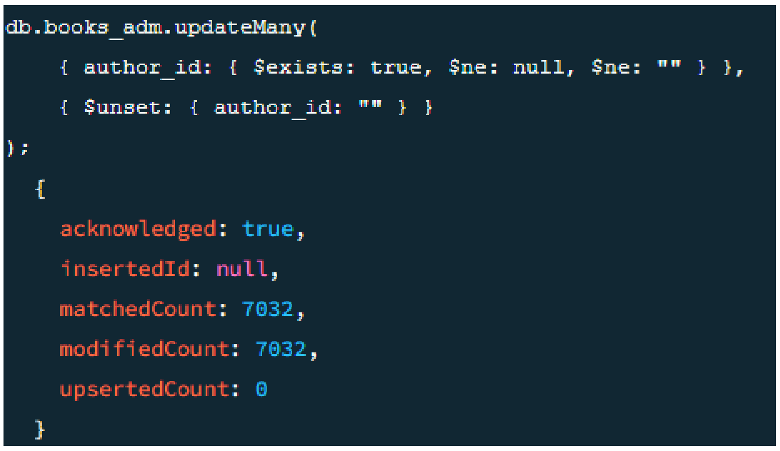
```
   _id: "6424452102decca634ec5ee2"
 ▸ genres: Array
   book_id: 1

   _id: "6424452102decca634ec5ee6"
 ▸ genres: Array
   book_id: 5

   _id: "6424452102decca634ec5ee8"
 ▸ genres: Array
   book_id: 7
```

After:



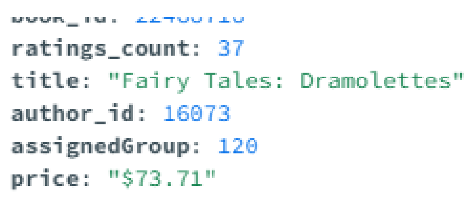
```
   _id: "6424452102decca634ec5ee2"
 ▸ genres: Array

   _id: "6424452102decca634ec5ee6"
 ▸ genres: Array

   _id: "6424452102decca634ec5ee8"
 ▸ genres: Array
```

Query Result: Book:



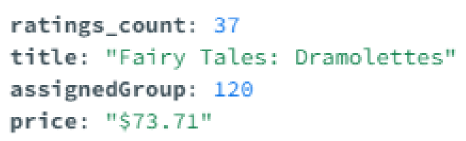
```
db.books_adm.updateMany(
    { author_id: { $exists: true, $ne: null, $ne: "" } },
    { $unset: { author_id: "" } }
);
  {
    acknowledged: true,
    insertedId: null,
    matchedCount: 7032,
    modifiedCount: 7032,
    upsertedCount: 0
  }
```

Before:



```
[illegible]
ratings_count: 37
title: "Fairy Tales: Dramolettes"
author_id: 16073
assignedGroup: 120
price: "$73.71"
```

After:

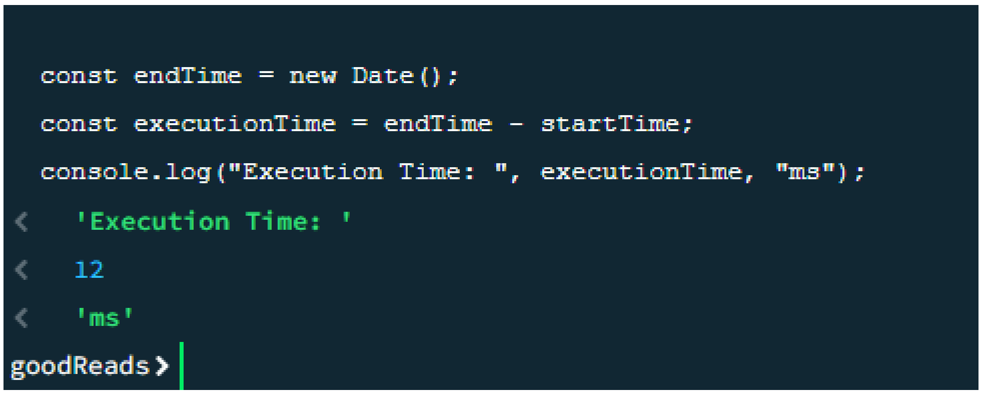


```
ratings_count: 37
title: "Fairy Tales: Dramolettes"
assignedGroup: 120
price: "$73.71"
```

5. Using $lookup operator, fetch the complete information of 50% of books with their authors and genres using pxxxxxxx_books collection. Track the speed of the query. [3 marks]

   Query:

```
///Tracking the time

 const startTime = new Date();

// Execute the query
db.books_adm.aggregate([
  { $sample: { size: Math.floor(db.books_adm.count() / 2) } },
  {
    $lookup: {
      from: "books_adm",
      localField: "author.author_id",
      foreignField: "author.author_id",
      as: "author_info"
    }
  },
  {
    $project: {
      _id: 0,
      isbn: 1,
      text_reviews_count: 1,
      series: 1,
      country_code: 1,
      is_ebook: 1,
      average_rating: 1,
      similar_books: 1,
      description: 1,
      format: 1,
      link: 1,
      authors: 1,
      publisher: 1,
      num_pages: 1,
      publication_day: 1,
      publication_month: 1,
      edition_information: 1,
      publication_year: 1,
      url: 1,
      image_url: 1,
      book_id: 1,
      ratings_count: 1,
      title: 1,
      assignedGroup: 1,
      price: 1,
      author: {
        $arrayElemAt: ["$author_info.author", 0]
        },
        genre: 1
      }
    }
]);

const endTime = new Date();
const executionTime = endTime - startTime;
console.log("Execution Time: ", executionTime, "ms");
```



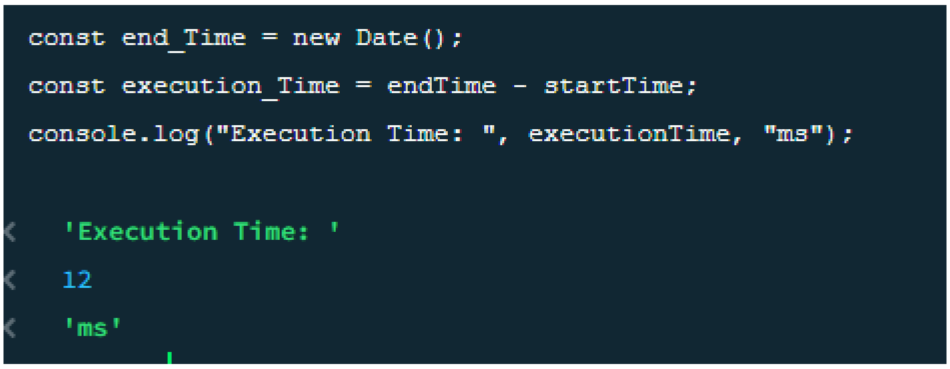
```
const endTime = new Date();
const executionTime = endTime - startTime;
console.log("Execution Time: ", executionTime, "ms");
<  'Execution Time: '
<  12
<  'ms'
goodReads>
```

Execution Time: 12ms.

6. Using the pxxxxxx_books_adm collection, fetch the same information as above. Track the speed of the query. [3 marks]

Question: fetch the complete information of 50% of books with their authors and genres. Track the speed of the query

```
const start_Time = new Date();

var totalDocuments = db.books_adm.find({ "author": { $exists: true } }).count();
var limit = Math.ceil(totalDocuments / 2);

db.books_adm.find({ "author": { $exists: true } }).limit(limit);


const end_Time = new Date();
const execution_Time = endTime - startTime;
console.log("Execution Time: ", executionTime, "ms");
```



```
const end_Time = new Date();
const execution_Time = endTime - startTime;
console.log("Execution Time: ", executionTime, "ms");

<   'Execution Time: '
<   12
<   'ms'
```

Execution Time: 12ms.

7. Compare the performance results of 7(5) & 7(6) above and write a brief discussion about the results. [4 marks]

Answer:

The performance results of the two queries indicate that they have similar execution times, both taking approximately 12 milliseconds. This suggests that the performance of the two queries is comparable.

The first query uses the aggregation framework with the **$sample** and **$lookup** stages to fetch a random sample of 50% of the books and perform a lookup to retrieve the author information. The second query uses the **find** method with the **$exists** operator and **limit** to fetch 50% of the books directly.

Both approaches are valid and can yield similar results. The choice between the two depends on the specific requirements and the structure of the data.

Overall, the execution time of 12 milliseconds for fetching 50% of the books is relatively fast, indicating efficient query execution. It's important to note that the actual performance may vary depending on the size of the collection, the complexity of the data, and the server resources.

.

8.0 WEEKLY JOURNALS [10 marks]

Report all your journal submissions by completing the table below:

Week No	Status/Reason
1	For example (Submitted on time)
2	For example (Late submission due to xyz)
3	For example (No submission due to abc)
4	
5	

9.0 DELIVERABLES

1. You are required to upload your answers to the questions into Turnitin on Blackboard as one PDF file. Do NOT submit any JSON file.

2. Add your P number to your filename.
3. Your file must be in a readable format so the NoSQL code in plain text can be copied and executed from it.
4. In Turnitin, press both the Upload button and the Confirm button to submit your file and receive a receipt.