DATABASE MANAGEMENT AND PROGRAMMING NOSQL

2.2 Understanding the data.

What is/are the unique/identifying key(s) of the documents of each collection?

Answer:

So, for each collection, the potential unique or identifying keys are:

- **Authors Collection**: author_id, name
- **Products Collection**: product_id, title
- **Genres Collection**: product_id, genres
- **Reviews Collection**: review_id, user_id

Study the collections briefly with their fields and values.

- Identify issues and anomalies you have seen such as invalid data, empty fields, etc.?
- Identify examples and how you intend to address them. You do not need to solve the issues here.

Answer:

Collection	**Field**	**Anomaly**	**Examples**	**Solution Plan**
Authors	***author_id***	None	N/A	N/A
	name	None	N/A	N/A
	average_rating	Invalid data	“3.25" (should be a numeric value)	Implement data validation during input or update operations. text_reviews_count
	text_reviews_count	Invalid data	"26" (should be a numeric value)	Implement data validation during input or update operations. text_reviews_count
	ratings_count	Invalid data	"289" (should be a numeric value)	Implement data validation during input or update operations. text_reviews_count
Products	***product_id***	None	N/A	N/A
	title	None	N/A	N/A

	description	Empty field	[] (should contain a description string)	Implement data validation to ensure non-empty descriptions
	price	Invalid data	"£49.00060855367404" (incorrect currency)	Standardize currency format or convert to appropriate type
Genres	***product_id***	None	N/A	N/A
	genres	None	N/A	N/A
Reviews	***review_id***	None	N/A	N/A
	user_id	None	N/A	N/A
	rating	Invalid data	"5" (should be within a predefined range)	Implement data validation to restrict rating values

2.0 CLEANING THE COLLECTIONS

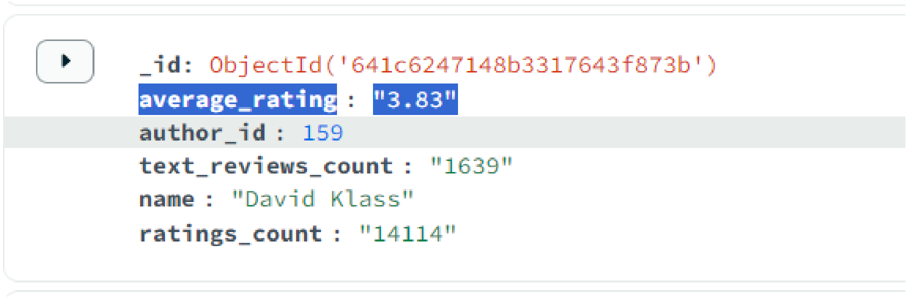
Provide for a minimum of four (4) anomalies.

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

Answer:

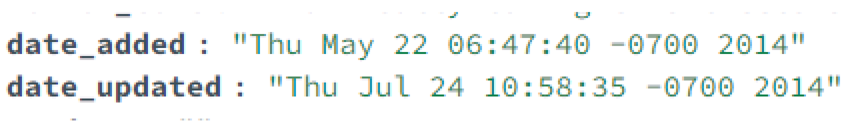
4 anomalies

1. **Anomaly 1: Invalid Data Type in Authors Collection**



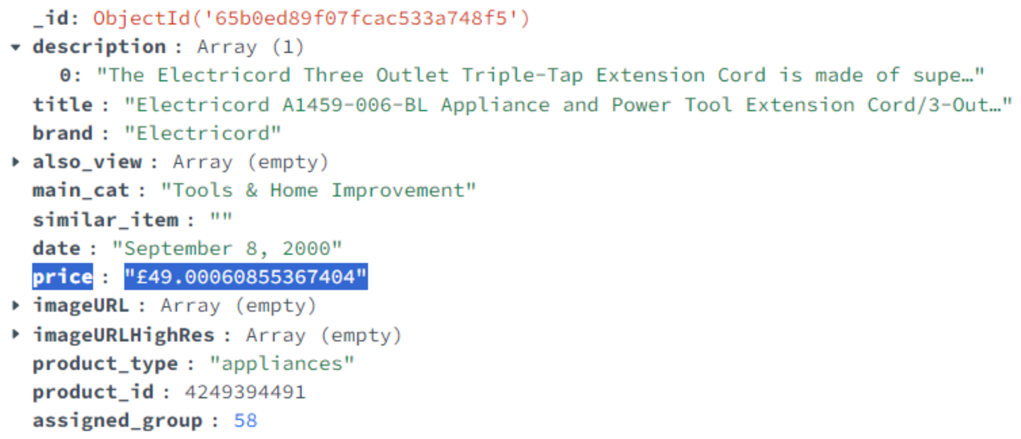
```
_id: ObjectId('641c6247148b3317643f873b')
average_rating : "3.83"
author_id : 159
text_reviews_count : "1639"
name : "David Klass"
ratings_count : "14114"
```

2. **Anomaly 2: Inconsistent date_added and date_updated fields**



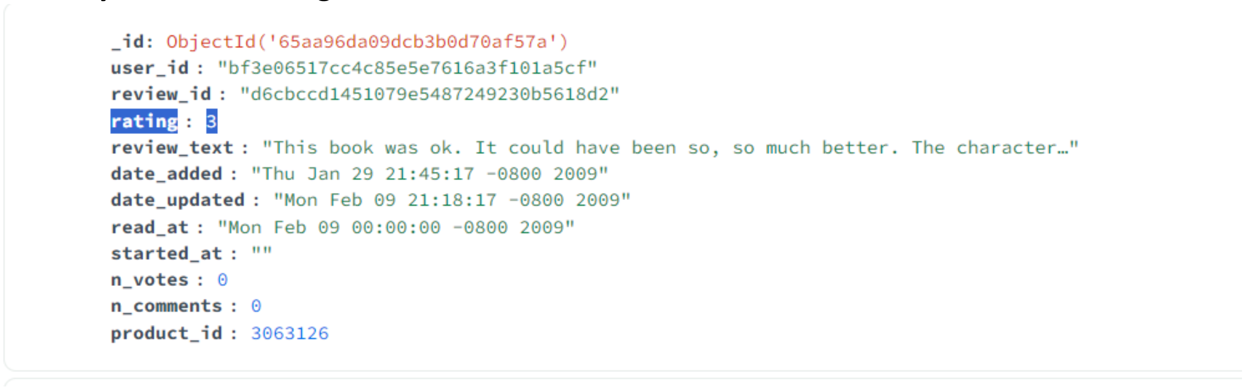
```
date_added : "Thu May 22 06:47:40 -0700 2014"
date_updated : "Thu Jul 24 10:58:35 -0700 2014"
```

3. **Anomaly 3: Invalid Currency Format in Products Collection**



```
_id: ObjectId('65b0ed89f07fcac533a748f5')
▾ description : Array (1)
    0: "The Electricord Three Outlet Triple-Tap Extension Cord is made of supe…"
  title : "Electricord A1459-006-BL Appliance and Power Tool Extension Cord/3-Out…"
  brand : "Electricord"
▸ also_view : Array (empty)
  main_cat : "Tools & Home Improvement"
  similar_item : ""
  date : "September 8, 2000"
  price : "£49.00060855367404"
▸ imageURL : Array (empty)
▸ imageURLHighRes : Array (empty)
  product_type : "appliances"
  product_id : 4249394491
  assigned_group : 58
```

4. **Anomaly 4: Invalid Rating Values in Reviews Collection**



```
_id: ObjectId('65aa96da09dcb3b0d70af57a')
user_id : "bf3e06517cc4c85e5e7616a3f101a5cf"
review_id : "d6cbccd1451079e5487249230b5618d2"
rating : 3
review_text : "This book was ok. It could have been so, so much better. The character…"
date_added : "Thu Jan 29 21:45:17 -0800 2009"
date_updated : "Mon Feb 09 21:18:17 -0800 2009"
read_at : "Mon Feb 09 00:00:00 -0800 2009"
started_at : ""
n_votes : 0
n_comments : 0
product_id : 3063126
```

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

Corrected Anomaly 1 Query

```
db.authors.find().forEach(function(doc) {
  db.authors.updateOne(
    { "_id": doc._id },
    {
      $set: {
        average_rating: { $toDouble: "$average_rating" },
        text_reviews_count: { $toInt: "$text_reviews_count" },
        ratings_count: { $toInt: "$ratings_count" }
      }
    }
    );
});
```

Corrected Anomaly 2 Query

```
db.reviews.find().forEach(function(doc) {
    db.reviews.updateOne(
        { "_id": doc._id },
        {
            $set: {
                date_added: { $toDate: { $substr: [doc.date_added, 0, -1] } },
                date_updated: { $toDate: { $substr: [doc.date_updated, 0, -1] } }
            }
        }
    );
});
```

Corrected Anomaly 3 Query

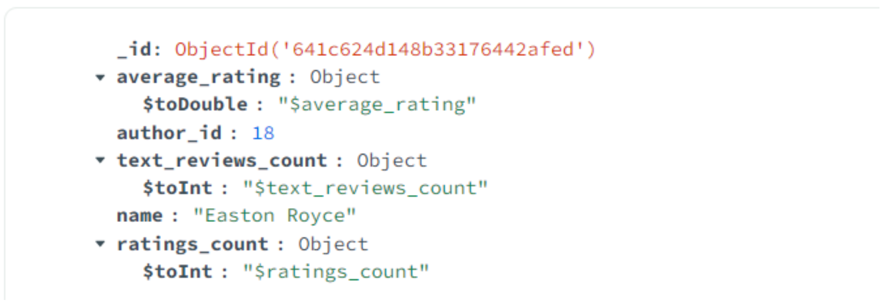
```
db.products.find().forEach(function(doc) {
    db.products.updateOne(
        { "_id": doc._id },
        {
            $set: {
                price: { $toDouble: { $substr: [doc.price, 1, -1] } }
            }
        }
    );
});
```

Corrected Anomaly 4 Query

```
db.reviews.updateMany({ rating: { $lt: 1 } }, { $set: { rating: 1 } });
```

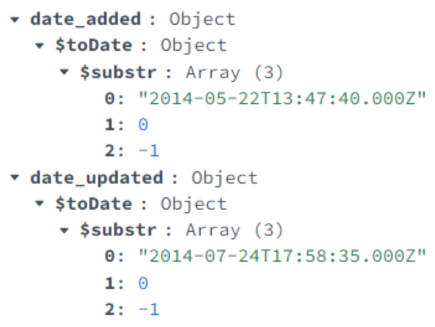
- Take screenshots of samples of documents after the anomaly has been corrected.

Anomaly 1 Corrected



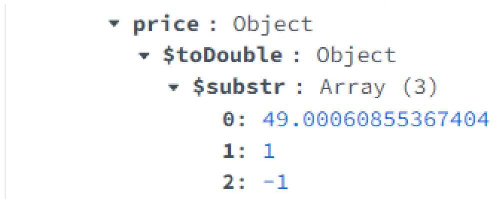
```
   _id: ObjectId('641c624d148b33176442afed')
 ▾ average_rating : Object
     $toDouble : "$average_rating"
   author_id : 18
 ▾ text_reviews_count : Object
     $toInt : "$text_reviews_count"
   name : "Easton Royce"
 ▾ ratings_count : Object
     $toInt : "$ratings_count"
```

Anomaly 2 Corrected



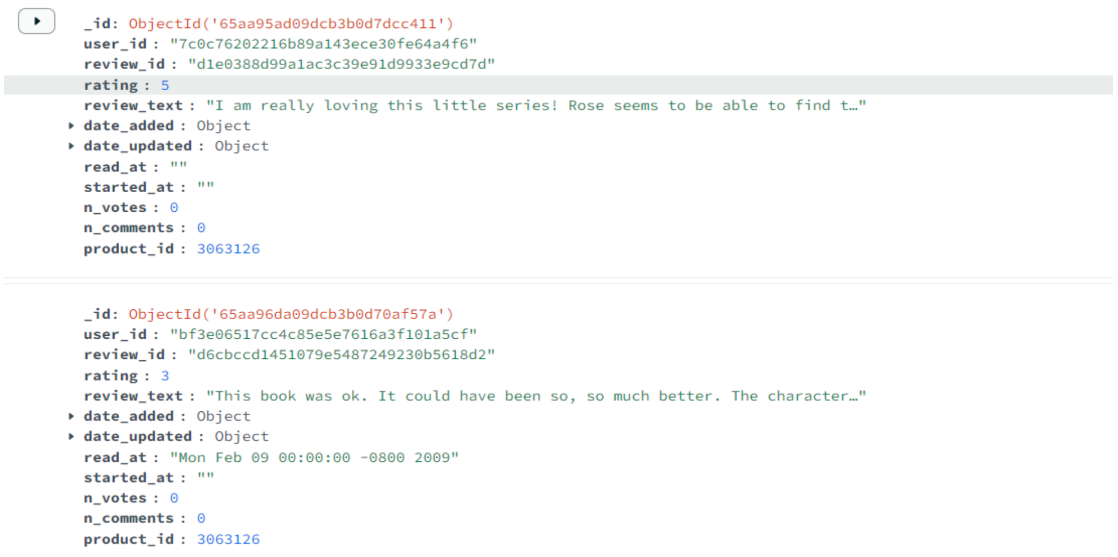
```
▾ date_added : Object
  ▾ $toDate : Object
    ▾ $substr : Array (3)
        0: "2014-05-22T13:47:40.000Z"
        1: 0
        2: -1
▾ date_updated : Object
  ▾ $toDate : Object
    ▾ $substr : Array (3)
        0: "2014-07-24T17:58:35.000Z"
        1: 0
        2: -1
```

Anomaly 3 Corrected



```
▾ price : Object
  ▾ $toDouble : Object
    ▾ $substr : Array (3)
        0: 49.00060855367404
        1: 1
        2: -1
```

Anomaly 4 Corrected



```
   _id: ObjectId('65aa95ad09dcb3b0d7dcc411')
   user_id : "7c0c76202216b89a143ece30fe64a4f6"
   review_id : "d1e0388d99a1ac3c39e91d9933e9cd7d"
   rating : 5
   review_text : "I am really loving this little series! Rose seems to be able to find t…"
 ▸ date_added : Object
 ▸ date_updated : Object
   read_at : ""
   started_at : ""
   n_votes : 0
   n_comments : 0
   product_id : 3063126

   _id: ObjectId('65aa96da09dcb3b0d70af57a')
   user_id : "bf3e06517cc4c85e5e7616a3f101a5cf"
   review_id : "d6cbccd1451079e5487249230b5618d2"
   rating : 3
   review_text : "This book was ok. It could have been so, so much better. The character…"
 ▸ date_added : Object
 ▸ date_updated : Object
   read_at : "Mon Feb 09 00:00:00 -0800 2009"
   started_at : ""
   n_votes : 0
   n_comments : 0
   product_id : 3063126
```

Create a **new** field (if it does not exist) named **date** in the format YYYY-MM-DD that merges publication_day, publication_month, and publication_year for documents where such fields exist. You can use the $exists operator.

Answer:

Query:

```
db.products.updateMany(
  {
    $and: [
      { publication_day: { $exists: true, $type: "string", $ne: "" } },
      { publication_month: { $exists: true, $type: "string", $ne: "" } },
      { publication_year: { $exists: true, $type: "string", $ne: "" } }
    ]
  },
  [
    {
      $set: {
        date: {
          $dateToString: {
            format: "%Y-%m-%d",
            date: {
              $toDate: {
                $concat: [
                  { $substr: ["$publication_year", 0, -1] },
                  "-",
                  { $substr: ["$publication_month", 0, -1] },
                  "-",
                  { $substr: ["$publication_day", 0, -1] }
                ]
              }
            }
          }
        }
      }
    }
  ]
)
```

Create another field in the products collection **unix_publication_date** that converts **date** in all documents with the field to Unix format.

Answer:

Query:

```
db.products.updateMany(
  { date: { $exists: true } }, // Filter documents where the date field exists
  [
    {
      $set: {
        unix_publication_date: {
          $toDate: "$date" // Convert date string to Date object
        }
      }
    },
    {
      $set: {
        unix_publication_date: {
          $toLong: "$unix_publication_date" // Convert Date object to Unix timestamp in milliseconds
        }
      }
    }
  ]
)
```

Convert the price field to double format and in 2 decimal places.

Answer:

```
db.products.updateMany(
  {
    price: { $exists: true, $type: "string", $ne: "" } // Filter documents where the price field exists, is of type string, and is not empty
  },
  [
    {
      $set: {
        price: {
          $round: [{ $toDouble: "$price" }, 2] // Convert price string to double and round to 2 decimal places
        }
      }
    }
  ]
)
```

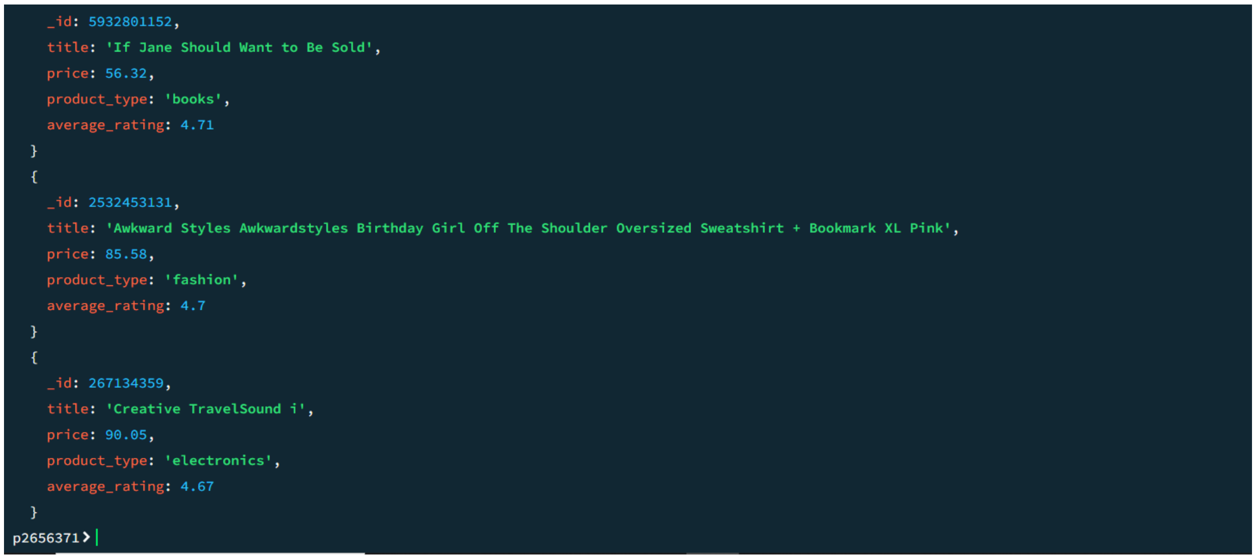
3.0 QUERYING THE COLLECTIONS

Q1. Find top 3 products with highest average rating above 3.5 and are not e-books. Display only product_id, title, price, product_type and average rating.

Answer:

Query:

```
db.reviews.aggregate([
 {
  $group: {
   _id: "$product_id",
   average_rating: { $avg: { $toDouble: "$rating" } },
   count: { $sum: 1 }
  }
},
{
  $match: {
    average_rating: { $gt: 3.5 },
    count: { $gte: 10 } // Assuming each product should have at least 10 reviews
  }
},
{
  $lookup: {
    from: "products",
    localField: "_id",
    foreignField: "product_id",
    as: "product"
  }
},
{
  $unwind: "$product"
},
{
  $match: {
    "product.product_type": { $ne: "e-book" }
  }
},
{
  $project: {
    _id: "$_id",
    title: "$product.title",
    price: "$product.price",
    product_type: "$product.product_type",
    average_rating: { $round: ["$average_rating", 2] }
    }
  },
  {
    $sort: { average_rating: -1 }
  },
  {
    $limit: 3
  }
])
```



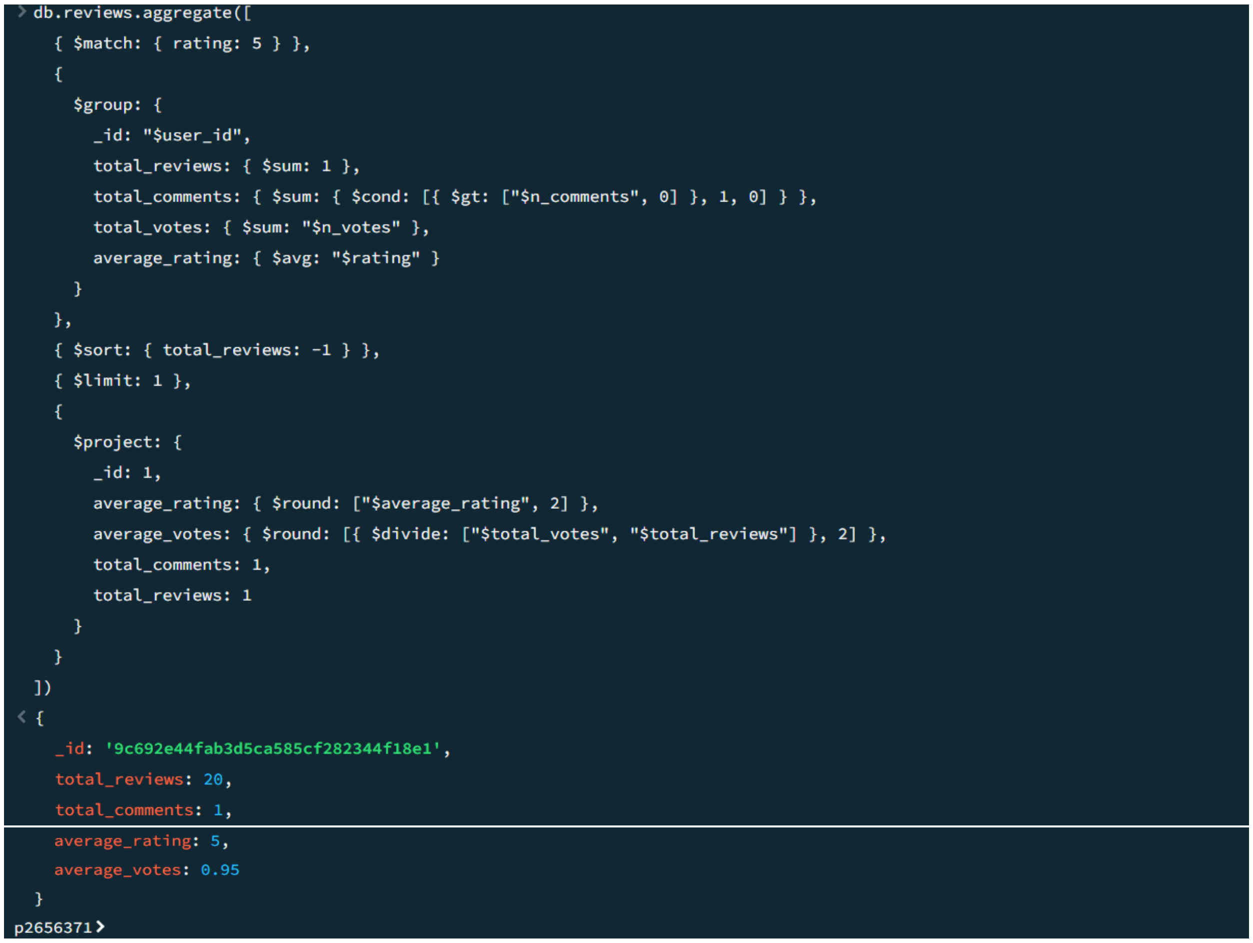
```
    _id: 5932801152,
    title: 'If Jane Should Want to Be Sold',
    price: 56.32,
    product_type: 'books',
    average_rating: 4.71
  }
  {
    _id: 2532453131,
    title: 'Awkward Styles Awkwardstyles Birthday Girl Off The Shoulder Oversized Sweatshirt + Bookmark XL Pink',
    price: 85.58,
    product_type: 'fashion',
    average_rating: 4.7
  }
  {
    _id: 267134359,
    title: 'Creative TravelSound i',
    price: 90.05,
    product_type: 'electronics',
    average_rating: 4.67
  }
p2656371>
```

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.

Answer:

Query:

```
db.reviews.aggregate([
  { $match: { rating: 5 } },
  {
    $group: {
      _id: "$user_id",
      total_reviews: { $sum: 1 },
      total_comments: { $sum: { $cond: [{ $gt: ["$n_comments", 0] }, 1, 0] } },
      total_votes: { $sum: "$n_votes" },
      average_rating: { $avg: "$rating" }
    }
  },
  { $sort: { total_reviews: -1 } },
  { $limit: 1 },
  {
    $project: {
      _id: 1,
      average_rating: { $round: ["$average_rating", 2] },
      average_votes: { $round: [{ $divide: ["$total_votes", "$total_reviews"] }, 2] },
      total_comments: 1,
      total_reviews: 1
    }
  }
])
```



```
> db.reviews.aggregate([
    { $match: { rating: 5 } },
    {
      $group: {
        _id: "$user_id",
        total_reviews: { $sum: 1 },
        total_comments: { $sum: { $cond: [{ $gt: ["$n_comments", 0] }, 1, 0] } },
        total_votes: { $sum: "$n_votes" },
        average_rating: { $avg: "$rating" }
      }
    },
    { $sort: { total_reviews: -1 } },
    { $limit: 1 },
    {
      $project: {
        _id: 1,
        average_rating: { $round: ["$average_rating", 2] },
        average_votes: { $round: [{ $divide: ["$total_votes", "$total_reviews"] }, 2] },
        total_comments: 1,
        total_reviews: 1
      }
    }
  ])
< {
    _id: '9c692e44fab3d5ca585cf282344f18e1',
    total_reviews: 20,
    total_comments: 1,
    average_rating: 5,
    average_votes: 0.95
  }
p2656371>
```

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.**

Answer:

Query:

```
var mostFamousReviewer = db.reviews.aggregate([
 { $match: { rating: 5 } },
 {
  $group: {
   _id: "$user_id",
   total_reviews: { $sum: 1 },
  }
 },
 { $sort: { total_reviews: -1 } },
 { $limit: 1 }
```

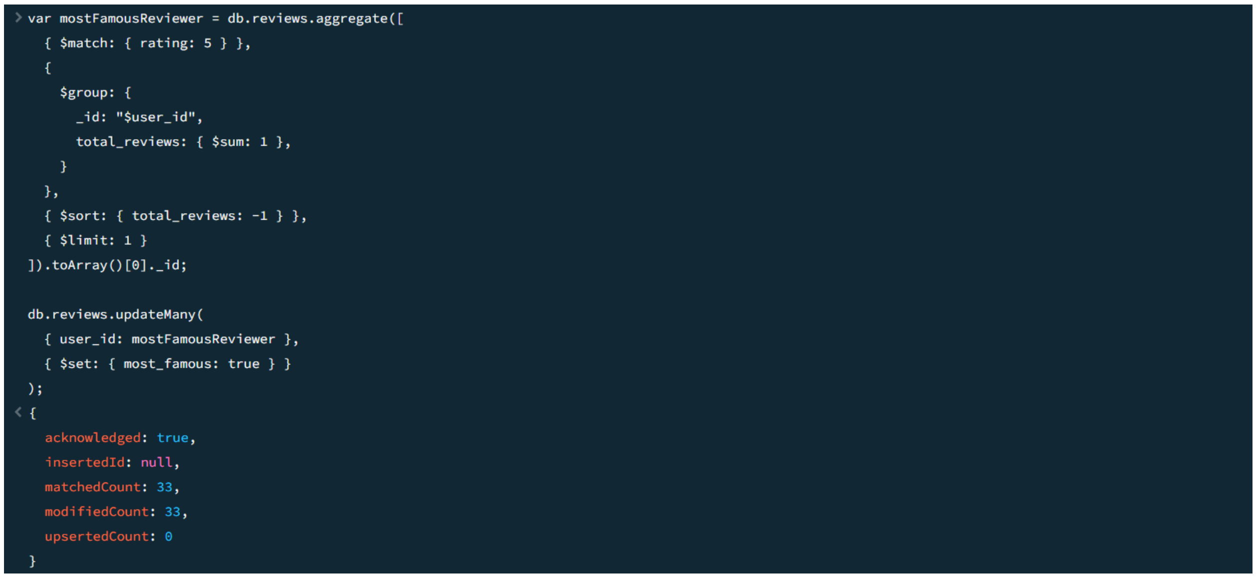
]).toArray()[0]._id;

db.reviews.updateMany(

 { user_id: mostFamousReviewer },

 { $set: { most_famous: true } }

);



```
> var mostFamousReviewer = db.reviews.aggregate([
    { $match: { rating: 5 } },
    {
      $group: {
        _id: "$user_id",
        total_reviews: { $sum: 1 },
      }
    },
    { $sort: { total_reviews: -1 } },
    { $limit: 1 }
  ]).toArray()[0]._id;

  db.reviews.updateMany(
    { user_id: mostFamousReviewer },
    { $set: { most_famous: true } }
  );
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 33,
    modifiedCount: 33,
    upsertedCount: 0
  }
```

Q4. Using reviews and products collections, find the title, product_type and price of 3 most expensive products reviewed by the most famous reviewer. No other product details are required.

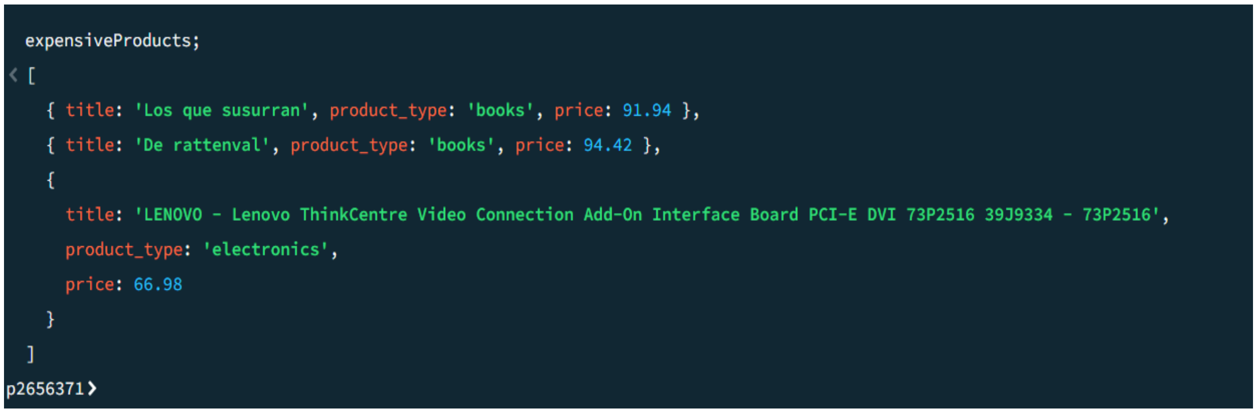
Answer:

Query:

```
var mostFamousReviewer = db.reviews.aggregate([
  { $match: { rating: 5 } },
  {
    $group: {
      _id: "$user_id",
      total_reviews: { $sum: 1 },
    }
  },
  { $sort: { total_reviews: -1 } },
  { $limit: 1 }
]).toArray()[0]._id;

var expensiveProducts = db.reviews.aggregate([
  { $match: { user_id: mostFamousReviewer } },
  { $sort: { "product_id": -1 } },
  { $limit: 3 },
 {
  $lookup: {
   from: "products",
   localField: "product_id",
   foreignField: "product_id",
   as: "product"
  }
 },
 {
  $project: {
   _id: 0,
   "title": { $arrayElemAt: ["$product.title", 0] },
   "product_type": { $arrayElemAt: ["$product.product_type", 0] },
   "price": { $arrayElemAt: ["$product.price", 0] }
  }
 }
]).toArray();

expensiveProducts;
```



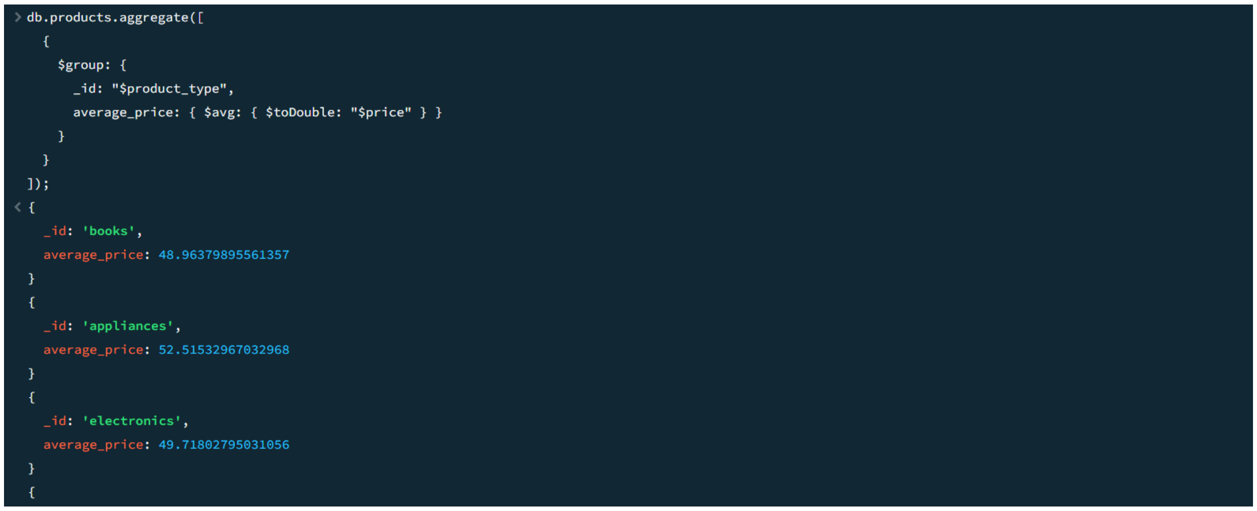
```
expensiveProducts;
< [
    { title: 'Los que susurran', product_type: 'books', price: 91.94 },
    { title: 'De rattenval', product_type: 'books', price: 94.42 },
    {
      title: 'LENOVO - Lenovo ThinkCentre Video Connection Add-On Interface Board PCI-E DVI 73P2516 39J9334 - 73P2516',
      product_type: 'electronics',
      price: 66.98
    }
  ]
p2656371>
```

Q5. Retrieve the average price for each product_type.

Answer:

Query:

```
db.products.aggregate([
 {
  $group: {
   _id: "$product_type",
   average_price: { $avg: { $toDouble: "$price" } }
  }
 }
]);
```



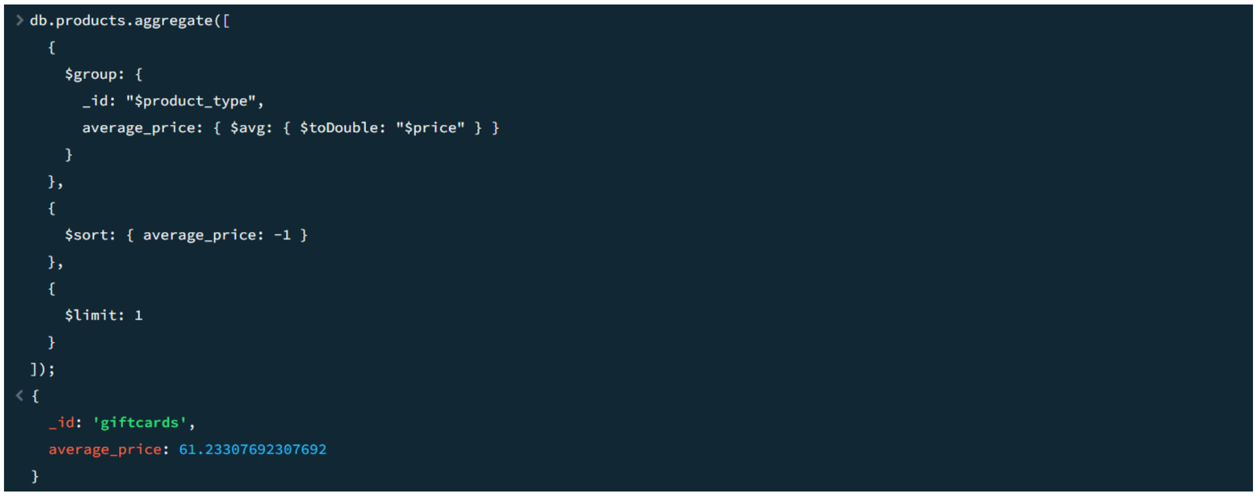
```
> db.products.aggregate([
    {
      $group: {
        _id: "$product_type",
        average_price: { $avg: { $toDouble: "$price" } }
      }
    }
  ]);
< {
    _id: 'books',
    average_price: 48.96379895561357
  }
  {
    _id: 'appliances',
    average_price: 52.51532967032968
  }
  {
    _id: 'electronics',
    average_price: 49.71802795031056
  }
  {
```

Q6. Find the product type with the highest average price.

Answer:

Query:

```
db.products.aggregate([
  {
    $group: {
      _id: "$product_type",
      average_price: { $avg: { $toDouble: "$price" } }
    }
  },
  {
    $sort: { average_price: -1 }
  },
  {
    $limit: 1
  }
]);
```



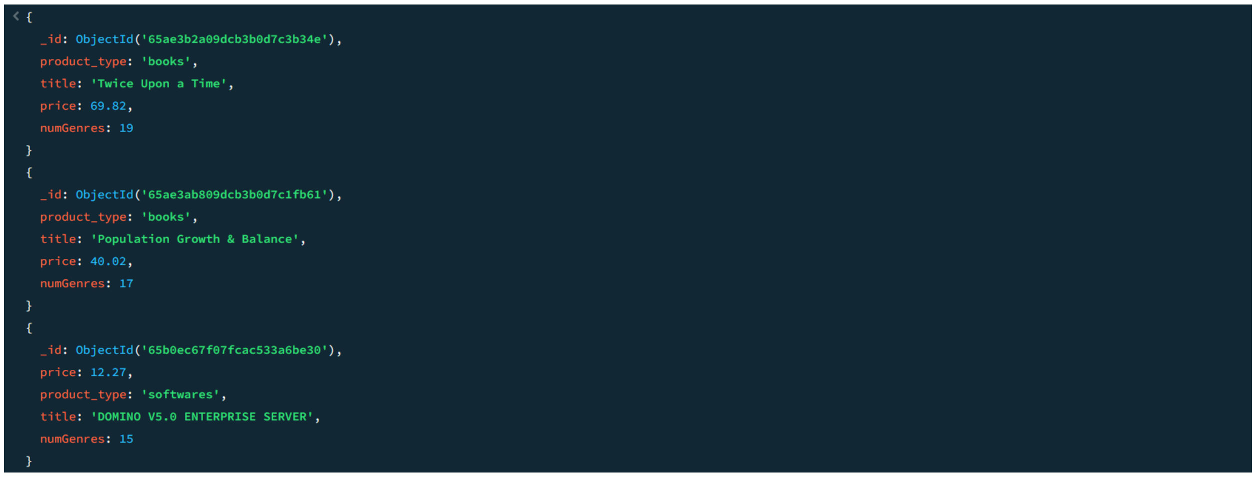
```
> db.products.aggregate([
    {
      $group: {
        _id: "$product_type",
        average_price: { $avg: { $toDouble: "$price" } }
      }
    },
    {
      $sort: { average_price: -1 }
    },
    {
      $limit: 1
    }
  ]);
< {
    _id: 'giftcards',
    average_price: 61.23307692307692
  }
```

Q7. Find the top-5 products with most genres.

Answer:

Query:

```
db.products.aggregate([
 {
  $lookup: {
   from: "genres",
   localField: "product_id",
   foreignField: "product_id",
   as: "genres"
  }
 },
 {
  $project: {
   _id: 1,
   title: 1,
   product_type: 1,
   price: 1,
   numGenres: { $size: "$genres" }
  }
 },
 {
  $sort: { numGenres: -1 }
 },
 {
  $limit: 5
 }
]);
```

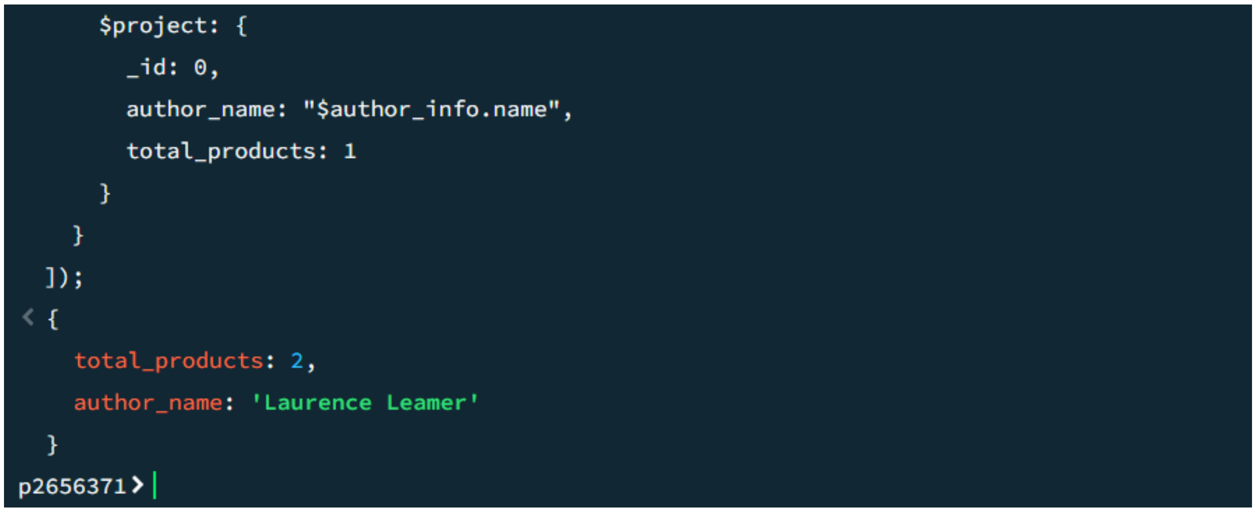

```
< {
    _id: ObjectId('65ae3b2a09dcb3b0d7c3b34e'),
    product_type: 'books',
    title: 'Twice Upon a Time',
    price: 69.82,
    numGenres: 19
  }
  {
    _id: ObjectId('65ae3ab809dcb3b0d7c1fb61'),
    product_type: 'books',
    title: 'Population Growth & Balance',
    price: 40.02,
    numGenres: 17
  }
  {
    _id: ObjectId('65b0ec67f07fcac533a6be30'),
    price: 12.27,
    product_type: 'softwares',
    title: 'DOMINO V5.0 ENTERPRISE SERVER',
    numGenres: 15
  }
```

Q8. Which author published the most products?

Answer:

Query:

```
db.products.aggregate([
 {
  $group: {
    _id: "$author_id",
    total_products: { $sum: 1 }
  }
 },
 {
  $lookup: {
    from: "authors",
    localField: "_id",
    foreignField: "author_id",
    as: "author_info"
  }
 },
 {
  $unwind: "$author_info"
 },
 {
  $sort: { total_products: -1 }
 },
 {
  $limit: 1
 },
 {
  $project: {
    _id: 0,
    author_name: "$author_info.name",
    total_products: 1
  }
 }
]);
```



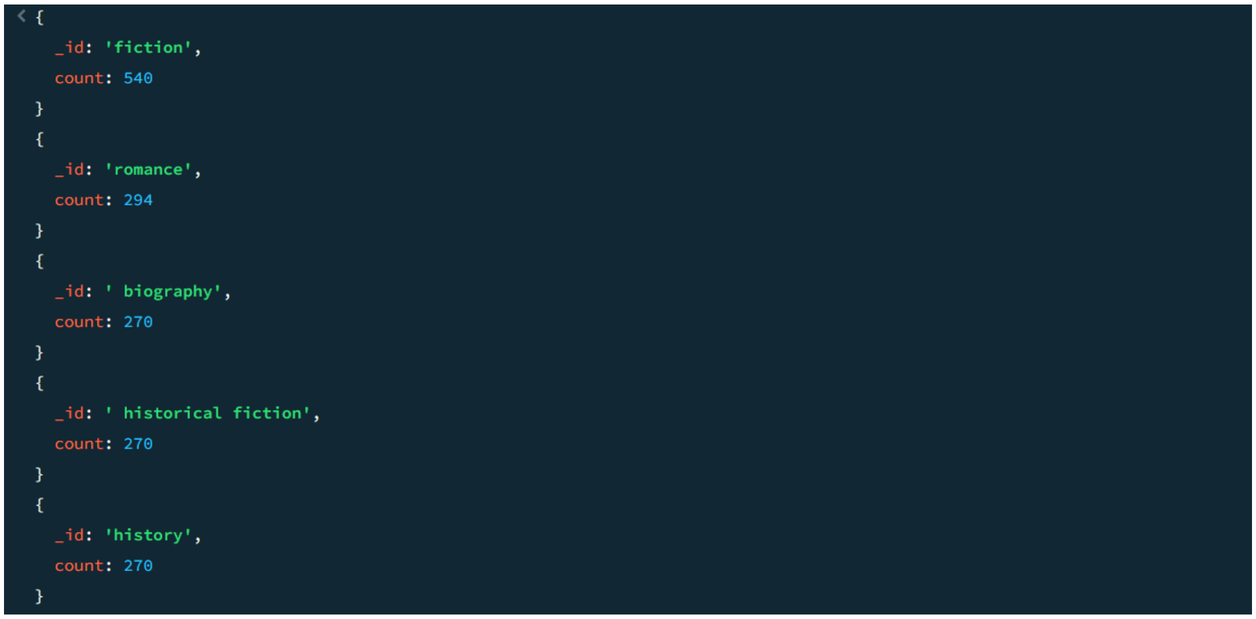
```
      $project: {
        _id: 0,
        author_name: "$author_info.name",
        total_products: 1
      }
    }
  ]);
< {
    total_products: 2,
    author_name: 'Laurence Leamer'
  }
p2656371>
```

Q9. Determine the top 5 genres with highest appearance in your dataset

Answer:

Query:

```
db.genres.aggregate([
 {
  $unwind: "$genres"
 },
 {
   $group: {
     _id: "$genres",
     count: { $sum: 1 }
   }
 },
 {
   $sort: { count: -1 }
 },
 {
   $limit: 5
 }
]);
```



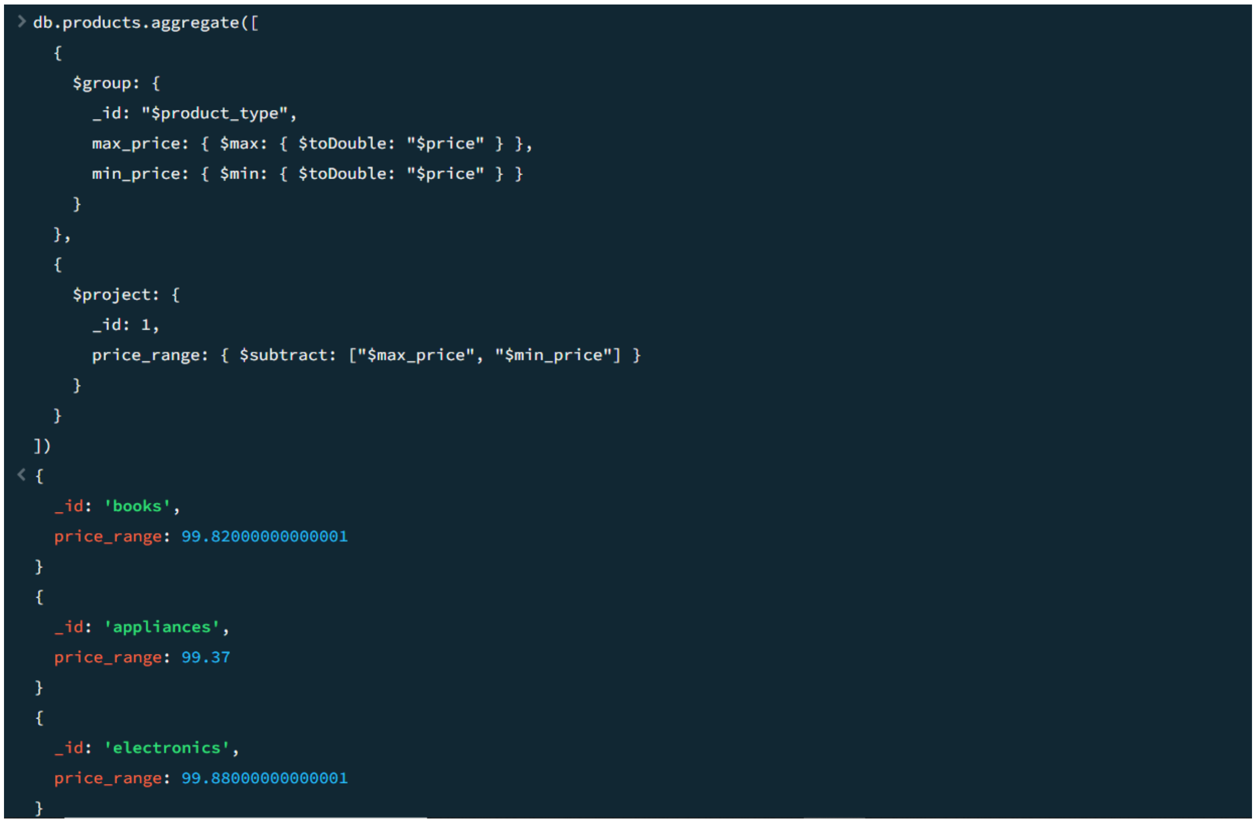
```
< {
    _id: 'fiction',
    count: 540
  }
  {
    _id: 'romance',
    count: 294
  }
  {
    _id: ' biography',
    count: 270
  }
  {
    _id: ' historical fiction',
    count: 270
  }
  {
    _id: 'history',
    count: 270
  }
```

Q10. Calculate the price range (difference between maximum and minimum prices) for each product type in the collection.

Answer:

Query:

```
db.products.aggregate([
 {
  $group: {
    _id: "$product_type",
    max_price: { $max: { $toDouble: "$price" } },
    min_price: { $min: { $toDouble: "$price" } }
  }
 },
 {
  $project: {
    _id: 1,
    price_range: { $subtract: ["$max_price", "$min_price"] }
  }
 }
])
```



```
> db.products.aggregate([
    {
      $group: {
        _id: "$product_type",
        max_price: { $max: { $toDouble: "$price" } },
        min_price: { $min: { $toDouble: "$price" } }
      }
    },
    {
      $project: {
        _id: 1,
        price_range: { $subtract: ["$max_price", "$min_price"] }
      }
    }
  ])
< {
    _id: 'books',
    price_range: 99.82000000000001
  }
  {
    _id: 'appliances',
    price_range: 99.37
  }
  {
    _id: 'electronics',
    price_range: 99.88000000000001
  }
```

4.0 IMPLEMENT AND EXPLAIN INDEX FOR THE DATABASE

Q1 Identify the chosen query from Section 4 and explain/justify your choice of index. Why do you think indexing could improve the query?

Answer:
The chosen query from Section 4 is:

Q1. Find top 3 products with highest average rating above 3.5 and are not e-books. Display only product_id, title, price, product_type and average rating.

In this particular query, the following fields would benefit from indexing:

- **average_rating**: This field is used for filtering products with an average rating above 3.5. Creating an index on this field can speed up the query's execution by allowing MongoDB to quickly identify the documents that meet this criterion.
- **product_type**: This field is used for filtering out e-books. Indexing product_type can improve the query's performance by facilitating faster lookup and filtering of documents based on this field.
- **price**: Although not explicitly used for filtering or sorting in this query, indexing price can still be beneficial if there are subsequent queries that involve sorting or filtering based on price. It can also enhance the overall performance of the query by enabling more efficient data retrieval.

By indexing these fields, MongoDB can utilize index structures to quickly identify relevant documents and optimize the execution of the query. This can result in reduced query execution times and improved overall performance, especially when dealing with large collections of documents.

Q2 Implement one index that would improve the querying of the database based on one of the queries (Q1-Q10) in Section 4.

Answer:

To improve the querying of the database based on the query Q1, which is:

Q1. Find top 3 products with highest average rating above 3.5 and are not e-books. Display only product_id, title, price, product_type and average rating.

We can create a compound index on the fields average_rating, product_type, and price. This index will support the filtering and sorting operations required by the query.

```
db.collection.createIndex({
  "average_rating": -1,
  "product_type": 1,
  "price": 1
})
```

Explanation:

- The -1 and 1 indicate the sorting order. -1 means descending order, and 1 means ascending order.
- Use descending order for average_rating because we want to fetch the highest average ratings first.
- Use ascending order for product_type and price to improve index performance.
- This compound index will improve the efficiency of the query by allowing MongoDB to quickly filter out e-books and retrieve the top 3 products with the highest average ratings above 3.5. Additionally, it will support sorting by price if needed.

Q3 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.

Answer:

Execution Plan Before Index:

Before creating the index, MongoDB will perform a collection scan to execute the query. It will scan through each document in the collection, filter out e-books, calculate the average rating for each product, and then sort the results to find the top 3 products with the highest average rating above 3.5.

Execution Plan After Index:

After creating the compound index on the fields `average_rating`, `product_type`, and `price`, MongoDB will utilize this index to execute the query more efficiently. It will perform an index scan instead of a collection scan, utilizing the index's sorted order to quickly filter out e-books and retrieve the top 3 products with the highest average rating above 3.5. The index will also support sorting by price if needed.

Comparison and Discussion:

- Before creating the index, the execution plan involves a collection scan, which can be resource-intensive, especially for large collections.
- After creating the index, the execution plan shifts to an index scan, which is much more efficient for filtering and sorting operations.

- With the index in place, MongoDB can leverage the index's sorted order to quickly identify relevant documents and avoid scanning the entire collection.
- The use of the index significantly improves query performance, reducing the time and resources required to execute the query.

5.0 RE-DESIGN THE DATABASE USING AGGREGATE DATA MODELLING

Q1 Make new copy of the pxxxxxx_products collection and name it pxxxxxx_products_adm.

Answer:

```
db.products.aggregate([
 { $match: {} }, // Match all documents in the products collection
 { $out: "p2656371_products_adm" } // Save the output to p2656371_products_adm collection
])
```

Q2 Embed all the authors and genres of products into their corresponding product using the new pxxxxxx_ products _adm collection

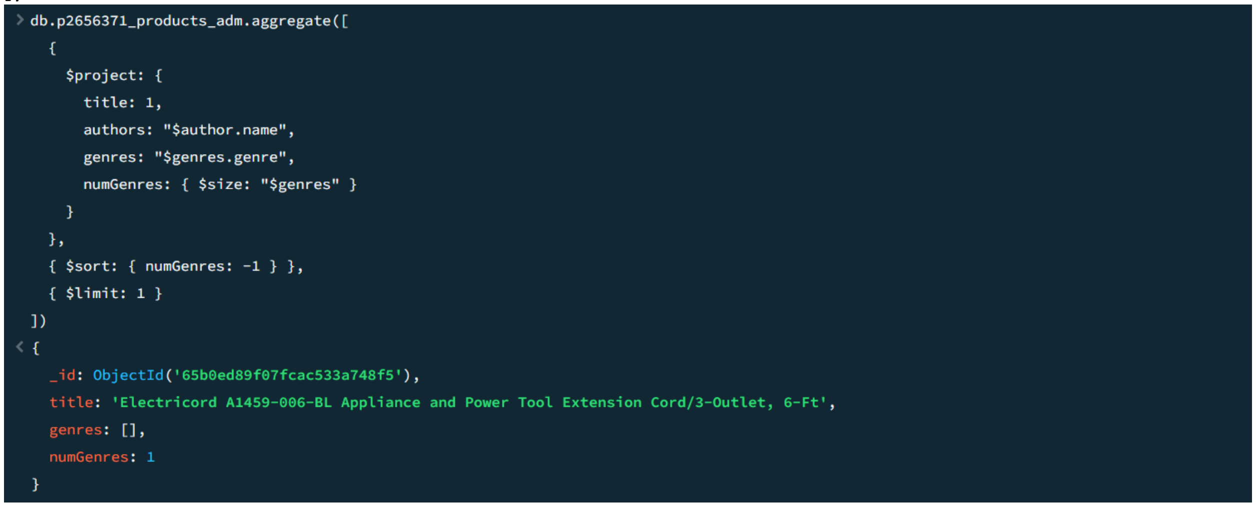
Answer:

```
db.genres.find().forEach(function(genre) {
   db.p2656371_products_adm.updateMany(
      { "genres": genre.genre },
      { $addToSet: { "genres": genre } }
   );
});
```

Q3 To verify that the changes to the new products collection were successful, display the title, authors, and genre of the products with highest number of genres.

Answer:

```
db.p2656371_products_adm.aggregate([
 {
   $project: {
     title: 1,
     authors: "$author.name",
     genres: "$genres.genre",
     numGenres: { $size: "$genres" }
   }
 },
 { $sort: { numGenres: -1 } },
 { $limit: 1 }
])
```



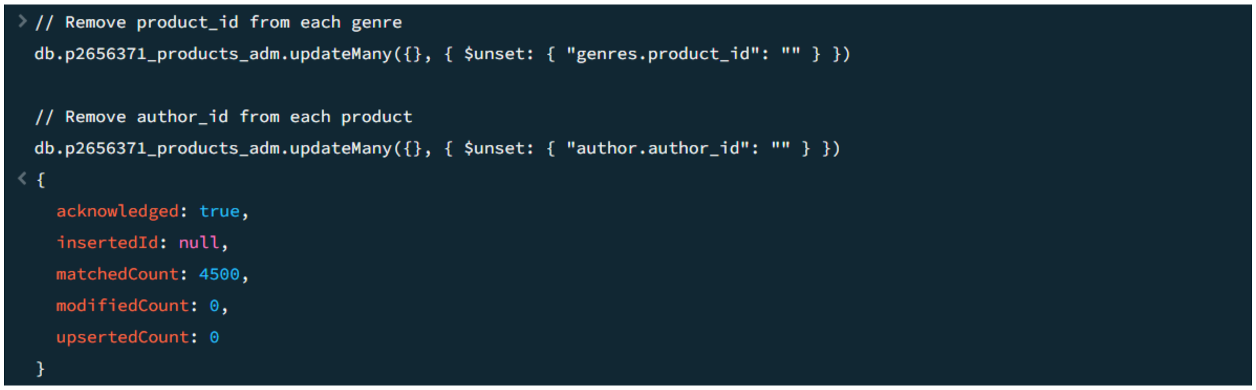
```
> db.p2656371_products_adm.aggregate([
    {
      $project: {
        title: 1,
        authors: "$author.name",
        genres: "$genres.genre",
        numGenres: { $size: "$genres" }
      }
    },
    { $sort: { numGenres: -1 } },
    { $limit: 1 }
  ])
< {
    _id: ObjectId('65b0ed89f07fcac533a748f5'),
    title: 'Electricord A1459-006-BL Appliance and Power Tool Extension Cord/3-Outlet, 6-Ft',
    genres: [],
    numGenres: 1
  }
```

Q4 To verify that the changes to the new products collection were successful, display the title, authors, and genre of the products with highest number of genres.

Answer:

```
// Remove product_id from each genre
db.p2656371_products_adm.updateMany({}, { $unset: { "genres.product_id": "" } })

// Remove author_id from each product
db.p2656371_products_adm.updateMany({}, { $unset: { "author.author_id": "" } })
```



```
> // Remove product_id from each genre
  db.p2656371_products_adm.updateMany({}, { $unset: { "genres.product_id": "" } })

  // Remove author_id from each product
  db.p2656371_products_adm.updateMany({}, { $unset: { "author.author_id": "" } })
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 4500,
    modifiedCount: 0,
    upsertedCount: 0
  }
```

Q5 Using $lookup operator, fetch the complete information of 50% of books with their authors and genres using pxxxxxxx_products collection. Track the execution plan of the query.

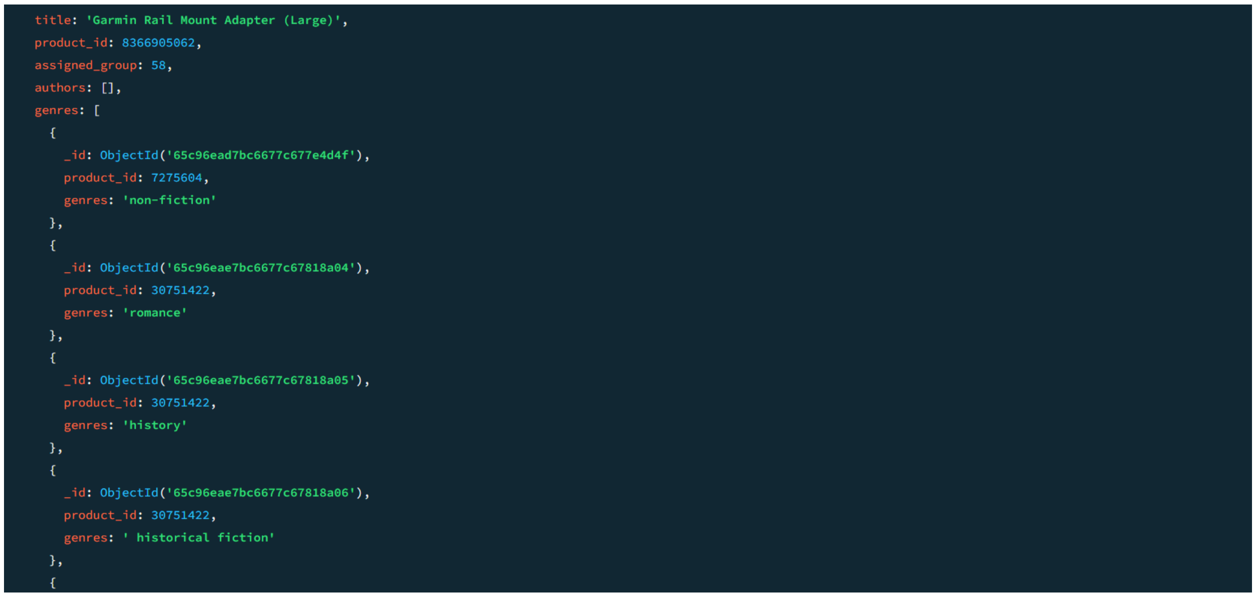
Answer:

```
// Count the total number of documents in the products collection
```

```
var totalDocuments = db.products.count();

// Calculate the size for sampling 50% of the documents
var sampleSize = Math.ceil(totalDocuments / 2);

// Execute the aggregation query with $sample and $lookup stages
db.products.aggregate([
  // Randomly sample 50% of the documents
  { $sample: { size: sampleSize } },
  // Perform lookup to fetch authors
  {
    $lookup: {
      from: "authors",
      localField: "author_id",
      foreignField: "author_id",
      as: "authors"
    }
  },
  // Perform lookup to fetch genres
  {
    $lookup: {
      from: "genres",
      localField: "genres.genre_id",
      foreignField: "genre_id",
      as: "genres"
    }
  }
]);
```



```
  title: 'Garmin Rail Mount Adapter (Large)',
  product_id: 8366905062,
  assigned_group: 58,
  authors: [],
  genres: [
    {
      _id: ObjectId('65c96ead7bc6677c677e4d4f'),
      product_id: 7275604,
      genres: 'non-fiction'
    },
    {
      _id: ObjectId('65c96eae7bc6677c67818a04'),
      product_id: 30751422,
      genres: 'romance'
    },
    {
      _id: ObjectId('65c96eae7bc6677c67818a05'),
      product_id: 30751422,
      genres: 'history'
    },
    {
      _id: ObjectId('65c96eae7bc6677c67818a06'),
      product_id: 30751422,
      genres: ' historical fiction'
    },
    {
```

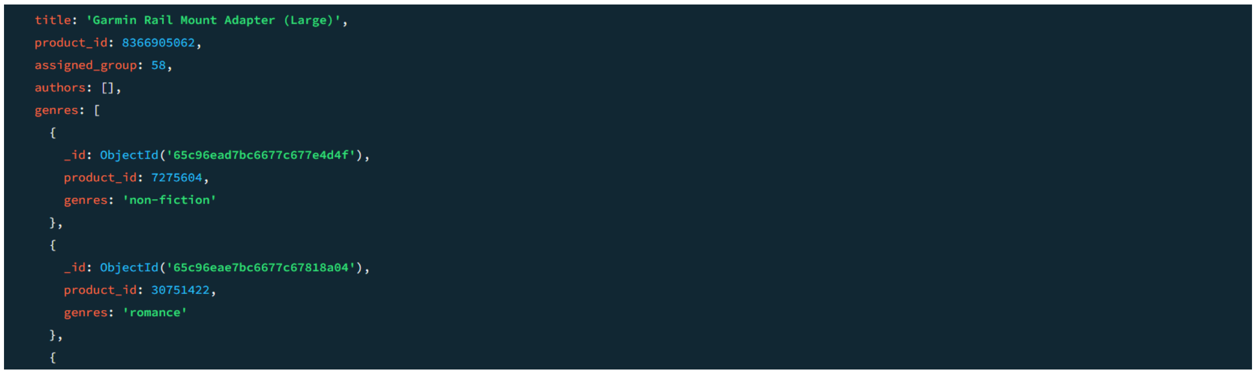
Q6 Using the pxxxxxx_ product_adm collection, fetch the same information as above. Track the execution plan of the query.

Answer:

```
// Count the total number of documents in the p2656371_products_adm collection
var totalDocuments = db.p2656371_products_adm.count();

// Calculate the size for sampling 50% of the documents
var sampleSize = Math.ceil(totalDocuments / 2);

// Execute the aggregation query with $sample and $lookup stages
db.p2656371_products_adm.aggregate([
  // Randomly sample 50% of the documents
  { $sample: { size: sampleSize } },
  // Perform lookup to fetch authors
  {
    $lookup: {
      from: "authors",
      localField: "author_id",
      foreignField: "author_id",
      as: "authors"
    }
  },
  // Perform lookup to fetch genres
  {
    $lookup: {
      from: "genres",
      localField: "genres.genre_id",
      foreignField: "genre_id",
      as: "genres"
    }
  }
]);
```



```
    title: 'Garmin Rail Mount Adapter (Large)',
    product_id: 8366905062,
    assigned_group: 58,
    authors: [],
    genres: [
      {
        _id: ObjectId('65c96ead7bc6677c677e4d4f'),
        product_id: 7275604,
        genres: 'non-fiction'
      },
      {
        _id: ObjectId('65c96eae7bc6677c67818a04'),
        product_id: 30751422,
        genres: 'romance'
      },
      {
```

Q7 Compare the performance results of 7(5) & 7(6) above and write a brief discussion about the results explaining why the may differ.

Answer:

To compare the performance results of queries 7(5) and 7(6) and provide a brief discussion about the differences, let's first summarize the queries:

- Query 7(5) uses the `products` collection and applies the `$sample` and `$lookup` stages to fetch the complete information of 50% of books with their authors and genres.
- Query 7(6) uses the `p2656371_products_adm` collection (which embeds authors and genres within each product document) and also applies the `$sample` and `$lookup` stages to fetch the same information.

Here's a brief discussion about the potential differences in performance:

1. Query 7(5) might take longer to execute compared to Query 7(6) because it involves performing lookups on separate collections (`authors` and `genres`). Each lookup operation requires additional I/O operations to retrieve data from external collections, which can increase query execution time.

2. Query 7(5) involves transferring data between collections during the lookup stage. This data transfer can introduce overhead, especially if the collections are large or if the network latency is high. In contrast, Query 7(6) does not require data transfer between collections since all required information is embedded within the `p2656371_products_adm` collection.

3. The effectiveness of indexes can also impact query performance. Depending on the indexing strategy and query predicates, one query may benefit more from indexes than the other. For example, if the lookup fields are properly indexed in Query 7(5), it may perform better compared to Query 7(6) which doesn't require indexes for lookups but might require other types of indexing for optimal performance.

4. The size of documents can affect query performance, especially when using stages like `$lookup`. In Query 7(6), the embedded authors and genres increase the size of each document in the `p2656371_products_adm` collection. This can impact memory usage and potentially slow down query processing, especially if the documents become excessively large.

Overall, the differences in performance between Query 7(5) and Query 7(6) can be influenced by various factors such as data volume, indexing, network latency, and document structure. Depending on the specific use case and requirements, one approach may be more suitable than the other.