POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Systems and Methods for Big and Unstructured Data Project

Author(s): **Francesca Pia Panaccione id: 10665920**
Francesco Santambrogio id: 10685653

Academic Year: 2023-2024

Contents

1 | Introduction

Medicines play a pivotal role in healthcare, serving as a cornerstone in the treatment, management, and prevention of diseases, illnesses, and various health conditions. Nowadays, medicines are mass-produced, and all of their meaningful information needs to be organized in some datasets. Such datasets, like the one used in this work, are massive, heterogeneous, and complex and therefore require big data solutions. In particular, a document database like MongoDB is suitable for this type of dataset since documents well represent objects with attributes, just like medicines with their description, cost, manufacturer information, and side effects. Some of these attributes are complex and are either expressed as other objects or arrays of attributes, both well handled by MongoDB via subdocuments and array type. MongoDB does not focus excessively on the relationships among objects, as graph databases do, but they are indeed poor in the present work's data set, nor on text search, as information search-based databases do, but it is not necessary with the existing attributes. Moreover, MongoDB uses a flexible document-based data model that allows for the storage of data in a schema-less manner and is designed for horizontal scalability by supporting sharding, namely allowing the distribution of data across multiple servers or clusters enabling high performance and scalability as data volumes grow: this is important in data sets like the one used in this work where there are hundreds of thousands of instances. It uses also indexing, query optimization, and native, efficient handling of JSON-like documents to enhance performance which is higher than a standard relational database, especially when dealing with huge datasets.

2 | Data Wrangling

After obtaining the dataset from its source, several data-wrangling steps were essential to cleanse and transform the data into an analyzable format. This aimed to generate dependable insights and streamline storage within a MongoDB database.

The primary emphasis was on restructuring or cleaning the values within the dataset's columns, outlined below:

1. Removal of currency symbols from the 'product_price' column and converting it into a numeric format. This was accomplished by executing the following code:

```
df['product_price'] = df['product_price'].replace({'': ''},
    regex=True).astype(float)
```

2. In the original dataset, the 'side_effects' column contained a string with effects separated by commas. To enhance query flexibility and streamline data organization, this string was split based on commas, resulting in the creation of a new column named 'side_effects_type.' This transformation aimed to convert the data into an array field, enabling more targeted queries for specific side effects. Additionally, this process facilitated essential grouping operations by structuring the side effects in a more accessible format

```
df['side_effects_type'] = df['side_effects'].apply(lambda x:
    [{'side_effect': effect} for effect in
    x.split(',')])updated_side_effects = [{'side_effect':
    [item['side_effect'] for item in row]} for row in
    df['side_effects_type']]

# Replace the 'side_effects_type' column with the updated values
df['side_effects_type'] = updated_side_effects

# Function to replace dictionary values with list values
```

```python
def replace_with_list(row):
    return row['side_effects_type']['side_effect']

# Apply the function to the DataFrame column
df['side_effects_type'] = df.apply(replace_with_list, axis=1)


# Converting the lists to EJSON format
df['side_effects_type'] = df['side_effects_type'].apply(lambda x:
    json.dumps(x))
```

3. The 'salt_composition' column initially combined both the chemical composition and corresponding doses within a single string. To enable separate queries for specific compositions or doses, the field structure was altered. It was transformed into a subdocument field featuring two distinct attributes: 'composition' and 'dose.' This transformation process involved employing regex-based operations to extract the pertinent information.

```python
# Extracting text within parentheses and creating a new column
df['salt_composition_dose'] =
    df['salt_composition'].str.extract(r'\((.*?)\)')

# Removing the extracted content from the 'salt_composition' column
df['salt_composition'] =
    df['salt_composition'].str.replace(r'\(.*?\)', '').str.strip()

# Function to process each row in the column
def transform_row(row):
    # Split the row based on '+'
    elements = [elem.strip() for elem in row.split('+')]
    return {"composition": elements}

# Apply the transformation to each row in the column
df['salt_composition'] = df['salt_composition'].apply(transform_row)


def modify_salt_composition(row):
    composition = row['salt_composition']['composition']
    dose = row['salt_composition_dose']
    row['salt_composition'] = {'composition': composition, 'dose':
        dose}
    return row

# Apply the function to the DataFrame
df = df.apply(modify_salt_composition, axis=1)

df['salt_composition'] = df['salt_composition'].apply(lambda x:
    json.dumps(x))
```

4. The transformation within the 'drug_interactions' column involved converting the initial single textual field into an array of subdocuments. Each subdocument comprised the following attributes:
   - 'drug': Represents the drug involved in the interaction.
   - 'brand': Corresponds to the brand of the interacting drug. An array containing effect lists. This process aimed to restructure the
   - 'effect': Specifies the level of severity of the interaction.

```
def process_interaction(interaction_string):
    interaction_data = json.loads(interaction_string)

    interactions = []
    min_length = min(len(interaction_data['drug']),
        len(interaction_data['brand']),
        len(interaction_data['effect']))

    for i in range(min_length):
        interaction = {
            'drug': interaction_data['drug'][i],
            'brand': interaction_data['brand'][i],
            'effect': interaction_data['effect'][i]
        }
        interactions.append(interaction)

    if len(interaction_data['drug']) !=
       len(interaction_data['brand']) or
       len(interaction_data['drug']) !=
       len(interaction_data['effect']):
        diff1 = len(interaction_data['drug']) - min_length
        diff2 = len(interaction_data['brand']) - min_length
        diff3 = len(interaction_data['effect']) - min_length

        for i in range(abs(diff1)):
            interactions.append({
                'drug': interaction_data['drug'][min_length + i] if
                   diff1 > 0 else 'nan',
                'brand': interaction_data['brand'][min_length + i] if
                   diff2 > 0 else 'nan',
                'effect': interaction_data['effect'][min_length + i]
                   if diff3 > 0 else 'nan'
            })

    return {"interactions": interactions}
```

5. The final refinement centered on the 'brand' field within the drug interaction column. Acknowledging the potential existence of multiple associated brands for a specific drug, the decision was made to replace the initial 'brand' attribute, which was a single text field, with an array encompassing all the values. By leveraging a comma-based split operation, this transformation resulted in an array containing diverse brand names linked with the drug, offering a more comprehensive and detailed representation of the data.

```
# Function to transform 'brand' column into an array
def transform_brand(brand):
    # If brand contains commas, split by commas and return a list
    if ',' in brand:
        brand_list = [b.strip() for b in brand.split(',')]
        return brand_list
    else:
        return [brand] # Return single value as a list

```

```python
# Function to process 'drug_interactions' column
def process_drug_interactions(interactions):
    # Convert string representation to a dictionary
    interactions_dict = json.loads(interactions)

    # Transform 'brand' within 'interactions' into an array
    for interaction in interactions_dict['interactions']:
        interaction['brand'] = transform_brand(interaction['brand'])

    # Convert back to Extended JSON (EJSON) format
    return json.dumps(interactions_dict)

# Apply the transformation to the 'drug_interactions' column
df['drug_interactions'] =
    df['drug_interactions'].apply(process_drug_interactions)
```

3 | Dataset

The dataset used in this work contains information about medications, in particular their descriptions, compositions, drug interactions, side effects, prices, manufacturers, and related details. The dataset contains only medicines used in India, with consequent prices expressed in Indian Rupee and Indian manufacturers.
Its non-relational schema is the following:

```
{
  "_id": ObjectId,
  "sub_category": String,
  "product_name": String,
  "salt_composition": Object {
    "composition": Array-[String],
    "dose": String
  },
  "product_price": Float,
  "product_manufactured": String,
  "medicine_desc": String,
  "drug_interactions": Array-[Object] [
    {
      "drug": String,
      "brand": Array-[String],
      "effect": String
    }
  ],
  "side_effects_type": Array-[String]
}
```

The structure of the data set is the following: Principle document

- _id (ObjectId): Unique identifier for the medication
- sub_category (String): Sub-category of the medication
- product_name (String): Name of the medication
- salt_composition (Object): subdocument containing information about the salt composition of the medication
- product_price (Float): Price of the medication
- product_manufactured (String): Name of the manufacturer
- medicine_desc (String): Description of the medication
- interactions (Array of Objects): Array of subdocuments containing information about the interactions with the medication
- side_effects_type (Array of Strings): List of side effects associated with the medication

salt_composition subdocument

- composition (Array of Strings): List of components in the medication
- dose (String): Dosage information.

interaction subdocument

- drug (String): Name of the drug involved in the interaction
- brand (Array of String): List of brand names of the drug
- effect (String): Description of the effect caused by the drug interaction

4 | Queries

For each performed query, a title, a description of what the query is supposed to do, the query itself, and its outcome are provided.

4.0.1. Query_1

Title: Filtered medications by salt composition doses

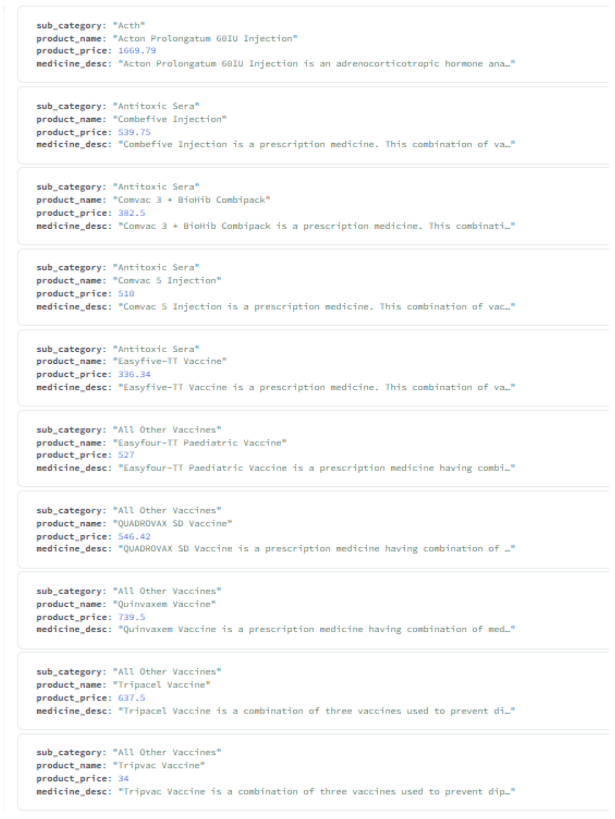
Description: This query aims to retrieve information about medicines that have a defined price and a specified dose ('30IU' or '60IU'). It then extracts specific fields from matching documents and arranges the results alphabetically by the product names. In particular, the steps are:

- Conditions: using the first parameter of the find() method
  - Price: The query filters for documents where the 'product_price' field exists.
  - Salt composition dose: The query further filters for documents where the 'salt_composition.dose' field matches either '30IU' or '60IU'
- Projection: using the second parameter of the find() method, including
  - 'product_name
  - 'sub_category'
  - 'medicine_desc'
  - 'product_price'
  - '_id' explicitly excluded
- Sorting: in ascending order based on the 'product_name' field

Query:

```
db.getCollection('medicine_data')
  .find(
    {
      product_price: { $exists: true },
      'salt_composition.dose': {
        $in: ['30IU', '60IU']
      }
    },
    {
      product_name: 1,
      sub_category: 1,
      medicine_desc: 1,
      product_price: 1,
      _id: 0
    }
  )
  .sort({ product_name: 1 });
```

Query outcome:



```
sub_category: "Acth"
product_name: "Acton Prolongatum 60IU Injection"
product_price: 1669.79
medicine_desc: "Acton Prolongatum 60IU Injection is an adrenocorticotropic hormone ana…"

sub_category: "Antitoxic Sera"
product_name: "Combefive Injection"
product_price: 539.75
medicine_desc: "Combefive Injection is a prescription medicine. This combination of va…"

sub_category: "Antitoxic Sera"
product_name: "Comvac 3 + BioHib Combipack"
product_price: 382.5
medicine_desc: "Comvac 3 + BioHib Combipack is a prescription medicine. This combinati…"

sub_category: "Antitoxic Sera"
product_name: "Comvac 5 Injection"
product_price: 510
medicine_desc: "Comvac 5 Injection is a prescription medicine. This combination of vac…"

sub_category: "Antitoxic Sera"
product_name: "Easyfive-TT Vaccine"
product_price: 336.34
medicine_desc: "Easyfive-TT Vaccine is a prescription medicine. This combination of va…"

sub_category: "All Other Vaccines"
product_name: "Easyfour-TT Paediatric Vaccine"
product_price: 527
medicine_desc: "Easyfour-TT Paediatric Vaccine is a prescription medicine having combi…"

sub_category: "All Other Vaccines"
product_name: "QUADROVAX SD Vaccine"
product_price: 546.42
medicine_desc: "QUADROVAX SD Vaccine is a prescription medicine having combination of …"

sub_category: "All Other Vaccines"
product_name: "Quinvaxem Vaccine"
product_price: 739.5
medicine_desc: "Quinvaxem Vaccine is a prescription medicine having combination of med…"

sub_category: "All Other Vaccines"
product_name: "Tripacel Vaccine"
product_price: 637.5
medicine_desc: "Tripacel Vaccine is a combination of three vaccines used to prevent di…"

sub_category: "All Other Vaccines"
product_name: "Tripvac Vaccine"
product_price: 34
medicine_desc: "Tripvac Vaccine is a combination of three vaccines used to prevent dip…"
```

4.0.2. Query_2

Title: Filtered medications by salt composition and manufacturer exclusion

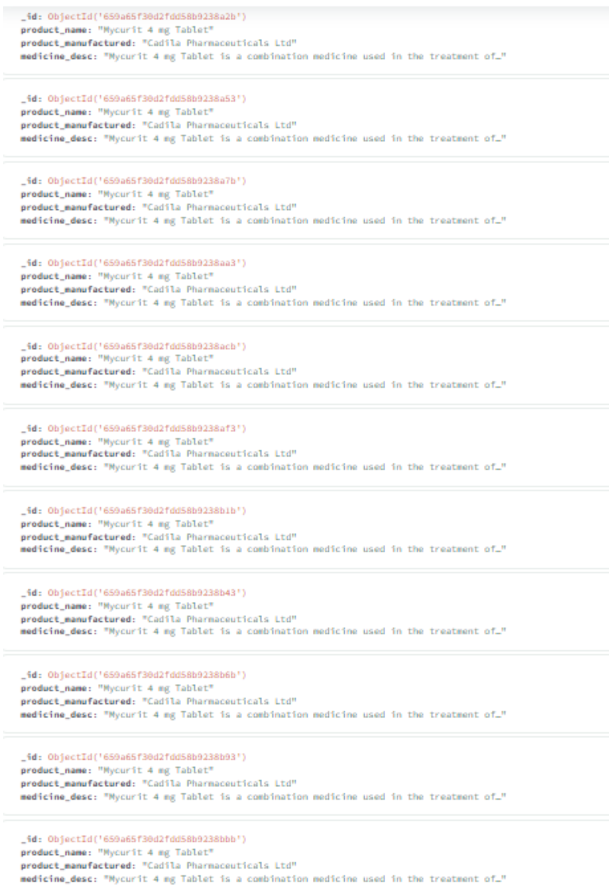
Description: This query aims to retrieve information about medications that contain a specific combination of salts (Isoniazid, Pyrazinamide, and Ethambutol) in their composition while excluding medicines manufactured by 'Lupin Ltd' and 'Macleods Pharmaceuticals Pvt Ltd'. It then fetches specific fields from matching documents, focusing on the product name, medicine description, and manufacturer details. In particular, the steps are:

- Conditions: using the first parameter of the find() method
  - Salt composition: It filters documents where the 'salt_composition.composition' field contains all three values: 'Isoniazid', 'Pyrazinamide', and 'Ethambutol'. The $all operator ensures that each of these components is present in the 'composition' array field.
  - Manufacturer: The query further filters based on the 'product_manufactured' field. It excludes documents where the 'product_manufactured' field does not match specific values using the $nin (not in) operator. In this case, it excludes medicines manufactured by 'Lupin Ltd' and 'Macleods Pharmaceuticals Pvt Ltd'
- Projection: using the second parameter of the find() method, including
  - 'product_name
  - 'medicine_desc'
  - 'product_manufactured'

Query:

```
db.getCollection('medicine_data').find(
  {
    'salt_composition.composition': {
      $all: [
        'Isoniazid',
        'Pyrazinamide',
        'Ethambutol'
      ]
    },
    product_manufactured: {
      $nin: [
        'Lupin Ltd',
        'Macleods Pharmaceuticals Pvt Ltd'
      ]
    }
  },
  {
    product_name: 1,
    medicine_desc: 1,
    product_manufactured: 1
  }
);
```

Query outcome:



```
_id: ObjectId('659a65f30d2fdd58b9238a2b')
product_name: "Mycurit 4 mg Tablet"
product_manufactured: "Cadila Pharmaceuticals Ltd"
medicine_desc: "Mycurit 4 mg Tablet is a combination medicine used in the treatment of…"

_id: ObjectId('659a65f30d2fdd58b9238a53')
product_name: "Mycurit 4 mg Tablet"
product_manufactured: "Cadila Pharmaceuticals Ltd"
medicine_desc: "Mycurit 4 mg Tablet is a combination medicine used in the treatment of…"

_id: ObjectId('659a65f30d2fdd58b9238a7b')
product_name: "Mycurit 4 mg Tablet"
product_manufactured: "Cadila Pharmaceuticals Ltd"
medicine_desc: "Mycurit 4 mg Tablet is a combination medicine used in the treatment of…"

_id: ObjectId('659a65f30d2fdd58b9238aa3')
product_name: "Mycurit 4 mg Tablet"
product_manufactured: "Cadila Pharmaceuticals Ltd"
medicine_desc: "Mycurit 4 mg Tablet is a combination medicine used in the treatment of…"

_id: ObjectId('659a65f30d2fdd58b9238acb')
product_name: "Mycurit 4 mg Tablet"
product_manufactured: "Cadila Pharmaceuticals Ltd"
medicine_desc: "Mycurit 4 mg Tablet is a combination medicine used in the treatment of…"

_id: ObjectId('659a65f30d2fdd58b9238af3')
product_name: "Mycurit 4 mg Tablet"
product_manufactured: "Cadila Pharmaceuticals Ltd"
medicine_desc: "Mycurit 4 mg Tablet is a combination medicine used in the treatment of…"

_id: ObjectId('659a65f30d2fdd58b9238b1b')
product_name: "Mycurit 4 mg Tablet"
product_manufactured: "Cadila Pharmaceuticals Ltd"
medicine_desc: "Mycurit 4 mg Tablet is a combination medicine used in the treatment of…"

_id: ObjectId('659a65f30d2fdd58b9238b43')
product_name: "Mycurit 4 mg Tablet"
product_manufactured: "Cadila Pharmaceuticals Ltd"
medicine_desc: "Mycurit 4 mg Tablet is a combination medicine used in the treatment of…"

_id: ObjectId('659a65f30d2fdd58b9238b6b')
product_name: "Mycurit 4 mg Tablet"
product_manufactured: "Cadila Pharmaceuticals Ltd"
medicine_desc: "Mycurit 4 mg Tablet is a combination medicine used in the treatment of…"

_id: ObjectId('659a65f30d2fdd58b9238b93')
product_name: "Mycurit 4 mg Tablet"
product_manufactured: "Cadila Pharmaceuticals Ltd"
medicine_desc: "Mycurit 4 mg Tablet is a combination medicine used in the treatment of…"

_id: ObjectId('659a65f30d2fdd58b9238bbb')
product_name: "Mycurit 4 mg Tablet"
product_manufactured: "Cadila Pharmaceuticals Ltd"
medicine_desc: "Mycurit 4 mg Tablet is a combination medicine used in the treatment of…"
```

4.0.3. Query_3

Title: Filtered medicines by dose and side effects count

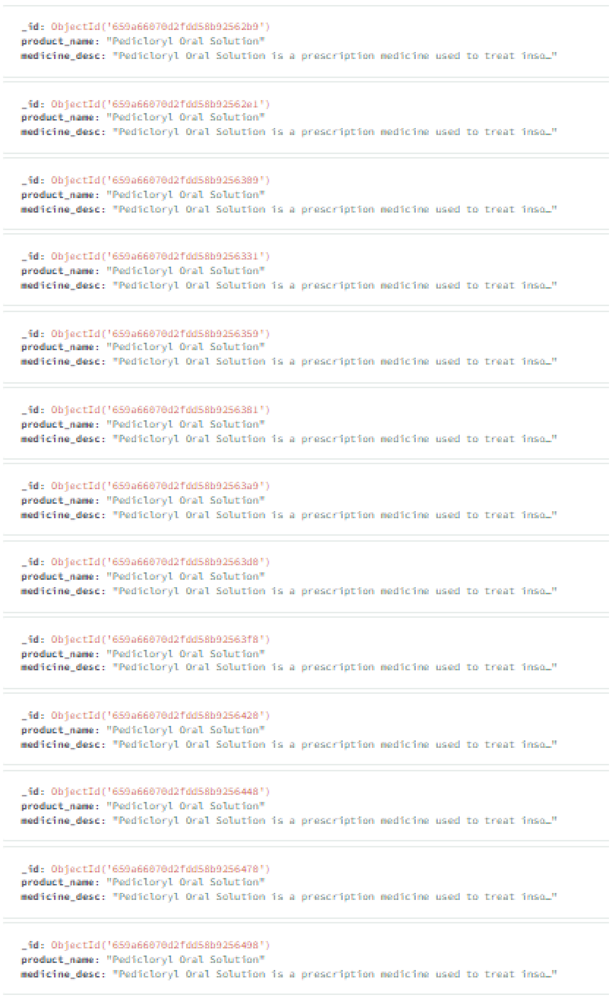
Description: This query aims to retrieve information about medications that have a specific salt composition dose ('500mg/5ml') and precisely four side effects listed in their data. It extracts the product name and medicine description from the matching documents. In particular, the steps are:

- Conditions: using the first parameter of the find() method
  - Logical AND Operator: The $and operator is utilized to specify that both conditions must be satisfied.
  - Salt Composition Dose: The query filters documents where the 'salt_composition.dose' field matches '500mg/5ml'
  - Side Effects Type: It further filters documents where the 'side_effects_type' array field has precisely 4 elements
- Projection: using the second parameter of the find() method, including
  - 'product_name
  - 'medicine_desc'

Query:

```
db.getCollection('medicine_data').find(
  {
    $and: [
      { 'salt_composition.dose': '500mg/5ml' },
      { side_effects_type: { $size: 4 } }
    ]
  },
  { product_name: 1, medicine_desc: 1}
);
```

Query outcome:



```
_id: ObjectId('659a66070d2fdd58b92562b9')
product_name: "Pedicloryl Oral Solution"
medicine_desc: "Pedicloryl Oral Solution is a prescription medicine used to treat inso…"

_id: ObjectId('659a66070d2fdd58b92562e1')
product_name: "Pedicloryl Oral Solution"
medicine_desc: "Pedicloryl Oral Solution is a prescription medicine used to treat inso…"

_id: ObjectId('659a66070d2fdd58b9256309')
product_name: "Pedicloryl Oral Solution"
medicine_desc: "Pedicloryl Oral Solution is a prescription medicine used to treat inso…"

_id: ObjectId('659a66070d2fdd58b9256331')
product_name: "Pedicloryl Oral Solution"
medicine_desc: "Pedicloryl Oral Solution is a prescription medicine used to treat inso…"

_id: ObjectId('659a66070d2fdd58b9256359')
product_name: "Pedicloryl Oral Solution"
medicine_desc: "Pedicloryl Oral Solution is a prescription medicine used to treat inso…"

_id: ObjectId('659a66070d2fdd58b9256381')
product_name: "Pedicloryl Oral Solution"
medicine_desc: "Pedicloryl Oral Solution is a prescription medicine used to treat inso…"

_id: ObjectId('659a66070d2fdd58b92563a9')
product_name: "Pedicloryl Oral Solution"
medicine_desc: "Pedicloryl Oral Solution is a prescription medicine used to treat inso…"

_id: ObjectId('659a66070d2fdd58b92563d8')
product_name: "Pedicloryl Oral Solution"
medicine_desc: "Pedicloryl Oral Solution is a prescription medicine used to treat inso…"

_id: ObjectId('659a66070d2fdd58b92563f8')
product_name: "Pedicloryl Oral Solution"
medicine_desc: "Pedicloryl Oral Solution is a prescription medicine used to treat inso…"

_id: ObjectId('659a66070d2fdd58b9256420')
product_name: "Pedicloryl Oral Solution"
medicine_desc: "Pedicloryl Oral Solution is a prescription medicine used to treat inso…"

_id: ObjectId('659a66070d2fdd58b9256448')
product_name: "Pedicloryl Oral Solution"
medicine_desc: "Pedicloryl Oral Solution is a prescription medicine used to treat inso…"

_id: ObjectId('659a66070d2fdd58b9256470')
product_name: "Pedicloryl Oral Solution"
medicine_desc: "Pedicloryl Oral Solution is a prescription medicine used to treat inso…"

_id: ObjectId('659a66070d2fdd58b9256498')
product_name: "Pedicloryl Oral Solution"
medicine_desc: "Pedicloryl Oral Solution is a prescription medicine used to treat inso…"
```

4.0.4. Query_4

Title: Medicines with filtered interactions, price, and subcategory

Description: This query aims to retrieve information about medications that meet specific criteria: having 'Alendronic Acid' and 'Aspirin' listed as interactions, a product price less than 100, and belonging to the 'Antacids Antiflatulents And Carminatives' subcategory. It retrieves selected fields and presents the medicine with the lowest price among matching documents. In particular, the steps are:

- Conditions: using the first parameter of the find() method
  - Logical AND Operator: Uses the $and operator to combine multiple conditions
  - Drug Interactions: Filters for documents where both 'Alendronic Acid' and 'Aspirin' are listed as interactions in the 'drug_interactions' array field

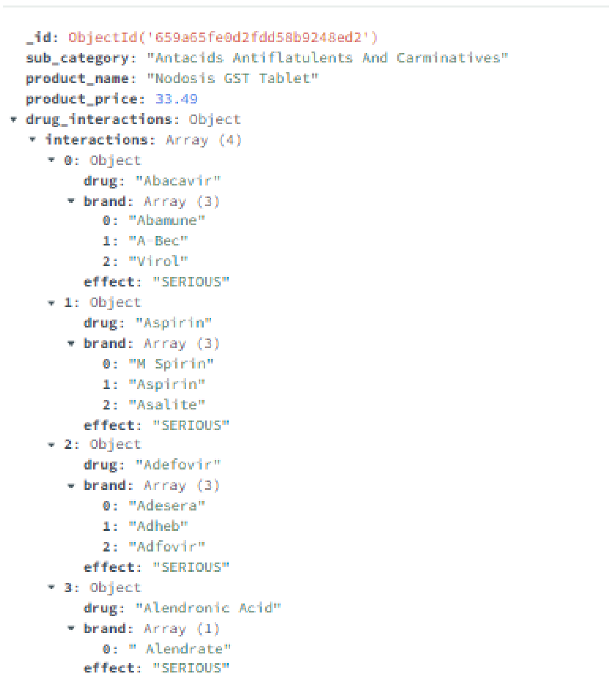
  - Product Price: Filters for documents where the 'product_price' exists and is less than 100.
  - Subcategory: Filters for documents in the 'Antacids Antiflatulents And Carminatives' subcategory.

- Projection: using the second parameter of the find() method, including
  - 'product_name
  - 'drug_interactions'
  - 'product_price'
  - 'sub_category'
- Sorting: in ascending order based on 'product_price'
- Limiting: just one for readability purposes

Query:

```
db.getCollection('medicine_data')
  .find(
    {
      $and: [
        {
          'drug_interactions.interactions.drug':
            'Alendronic Acid'
        },
        {
          'drug_interactions.interactions.drug':
            'Aspirin'
        },
        { product_price: { $lt: 100 } },
        { product_price: { $exists: true } },
        {
          sub_category:
            'Antacids Antiflatulents And Carminatives'
        }
      ]
    },
    {
      product_name: 1,
      drug_interactions: 1,
      product_price: 1,
      sub_category: 1
    }
  )
  .sort({ product_price: 1 })
  .limit(1);
```

Query outcome:



```
  _id: ObjectId('659a65fe0d2fdd58b9248ed2')
  sub_category: "Antacids Antiflatulents And Carminatives"
  product_name: "Nodosis GST Tablet"
  product_price: 33.49
▾ drug_interactions: Object
  ▾ interactions: Array (4)
    ▾ 0: Object
        drug: "Abacavir"
      ▾ brand: Array (3)
          0: "Abamune"
          1: "A-Bec"
          2: "Virol"
        effect: "SERIOUS"
    ▾ 1: Object
        drug: "Aspirin"
      ▾ brand: Array (3)
          0: "M Spirin"
          1: "Aspirin"
          2: "Asalite"
        effect: "SERIOUS"
    ▾ 2: Object
        drug: "Adefovir"
      ▾ brand: Array (3)
          0: "Adesera"
          1: "Adheb"
          2: "Adfovir"
        effect: "SERIOUS"
    ▾ 3: Object
        drug: "Alendronic Acid"
      ▾ brand: Array (1)
          0: " Alendrate"
        effect: "SERIOUS"
```

4.0.5. Query_5

Title: Filtered medicines by insulin subcategories and manufacturers and specific product exclusion

Description: This query aims to retrieve information about medications based on specific criteria, including different sub-categories of insulin products manufactured by different companies. It then excludes a specific product from the results and the selected fields are projected in the output. In particular, the steps are:

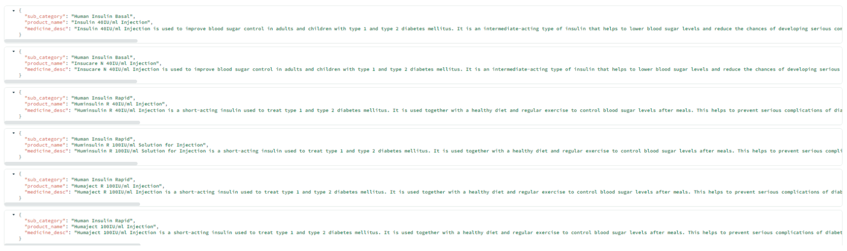
- Conditions: using the first parameter of the find() method
  - Logical AND Operator: Uses the $and operator to combine multiple conditions.

  - Nested OR Conditions: Two sets of conditions within $or, each representing a combination of sub-category and manufacturer: 'Human Insulin Basal' manufactured by 'Sun Pharmaceutical Industries Ltd', 'Human Insulin Rapid' manufactured by 'Eli Lilly and Company India Pvt Ltd'.
  - Product Exclusion: Uses the $nor operator to exclude documents where the 'product_name' field matches 'Huminsulin R 100IU Cartridge'
- Projection: using the second parameter of the find() method, including
  - 'sub_category'
  - 'product_name
  - 'medicine_desc'
  - '_id' explicitly excluded

Query:

```
db.getCollection('medicine_data').find(
  {
    $and: [
      {
        $or: [
          {
            $and: [
              {
                sub_category:
                  'Human Insulin Basal'
              },
              {
                product_manufactured:
                  'Sun Pharmaceutical Industries Ltd'
              }
            ]
          },
          {
            $and: [
              {
                sub_category:
                  'Human Insulin Rapid'
              },
              {
                product_manufactured:
                  'Eli Lilly and Company India Pvt Ltd'
              }
            ]
          }
        ]
      },
      {
        $nor: [
          {
            product_name:
              'Huminsulin R 100IU Cartridge'
          }
        ]
      }
    ]
  },
  {
    sub_category: 1,
    product_name: 1,
    medicine_desc: 1,
    _id: 0
  }
);
```

Query outcome:



```
▾ {
    "sub_category": "Human Insulin Basal",
    "product_name": "Insulin 40IU/ml Injection",
    "medicine_desc": "Insulin 40IU/ml Injection is used to improve blood sugar control in adults and children with type 1 and type 2 diabetes mellitus. It is an intermediate-acting type of insulin that helps to lower blood sugar levels and reduce the chances of developing serious com
  }
▾ {
    "sub_category": "Human Insulin Basal",
    "product_name": "Insucare N 40IU/ml Injection",
    "medicine_desc": "Insucare N 40IU/ml Injection is used to improve blood sugar control in adults and children with type 1 and type 2 diabetes mellitus. It is an intermediate-acting type of insulin that helps to lower blood sugar levels and reduce the chances of developing serious
  }
▾ {
    "sub_category": "Human Insulin Rapid",
    "product_name": "Huminsulin R 40IU/ml Injection",
    "medicine_desc": "Huminsulin R 40IU/ml Injection is a short-acting insulin used to treat type 1 and type 2 diabetes mellitus. It is used together with a healthy diet and regular exercise to control blood sugar levels after meals. This helps to prevent serious complications of dia
  }
▾ {
    "sub_category": "Human Insulin Rapid",
    "product_name": "Huminsulin R 100IU/ml Solution for Injection",
    "medicine_desc": "Huminsulin R 100IU/ml Solution for Injection is a short-acting insulin used to treat type 1 and type 2 diabetes mellitus. It is used together with a healthy diet and regular exercise to control blood sugar levels after meals. This helps to prevent serious compli
  }
▾ {
    "sub_category": "Human Insulin Rapid",
    "product_name": "Humaject R 100IU/ml Injection",
    "medicine_desc": "Humaject R 100IU/ml Injection is a short-acting insulin used to treat type 1 and type 2 diabetes mellitus. It is used together with a healthy diet and regular exercise to control blood sugar levels after meals. This helps to prevent serious complications of diab
  }
▾ {
    "sub_category": "Human Insulin Rapid",
    "product_name": "Humaject 100IU/ml Injection",
    "medicine_desc": "Humaject 100IU/ml Injection is a short-acting insulin used to treat type 1 and type 2 diabetes mellitus. It is used together with a healthy diet and regular exercise to control blood sugar levels after meals. This helps to prevent serious complications of diabet
  }
```

4.0.6. Query_6

Title: Medicines with a specific component and a serious drug interaction

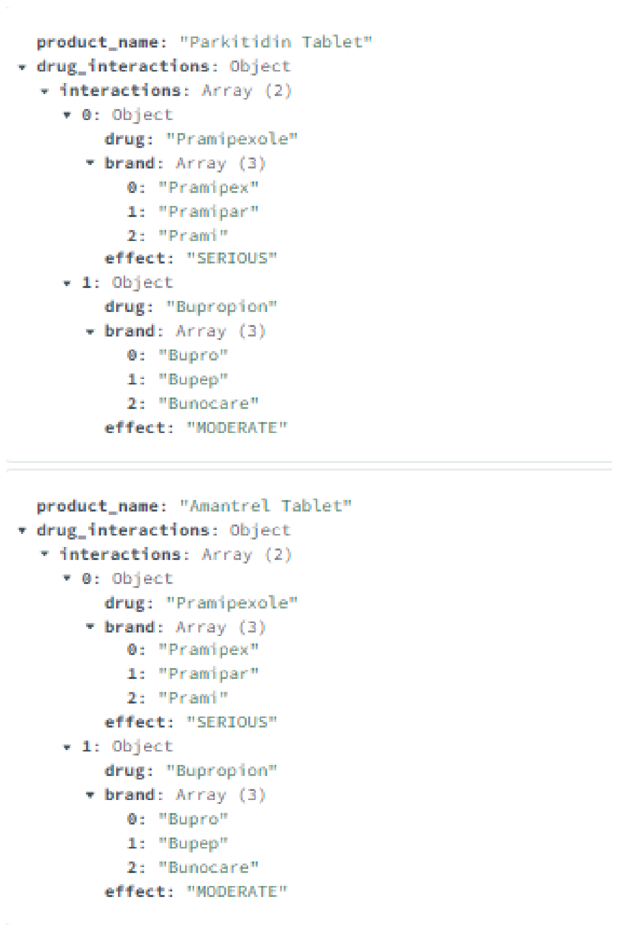
Description: This query aims to retrieve information about medications that have a serious drug interaction and contains 'Bupropion'. It fetches the product name and details about drug interactions from up to two matching documents. In particular, the steps are:

- Conditions: using the first parameter of the find() method
  - Drug Interactions: Filters for documents where there is a drug interaction defined as 'SERIOUS' and there is the component 'Bupropion'. This filters documents where a 'drug_interactions.interactions.effect' is 'SERIOUS' and a 'drug_interactions.interactions.drug' is 'Bupropion'
- Projection: using the second parameter of the find() method, including
  - 'product_name
  - 'drug_interactions'
  - '_id' explicitly excluded
- Limiting: just two for readability purposes

Query:

```
db.getCollection('medicine_data')
  .find(
    {
      'drug_interactions.interactions.effect':
        'SERIOUS',
      'drug_interactions.interactions.drug':
        'Bupropion'
    },
    {
      product_name: 1,
      drug_interactions: 1,
      _id: 0
    }
  )
  .limit(2);
```

Query outcome:



```
product_name: "Parkitidin Tablet"
▾ drug_interactions: Object
  ▾ interactions: Array (2)
    ▾ 0: Object
        drug: "Pramipexole"
      ▾ brand: Array (3)
          0: "Pramipex"
          1: "Pramipar"
          2: "Prami"
        effect: "SERIOUS"
    ▾ 1: Object
        drug: "Bupropion"
      ▾ brand: Array (3)
          0: "Bupro"
          1: "Bupep"
          2: "Bunocare"
        effect: "MODERATE"
```

```
product_name: "Amantrel Tablet"
▾ drug_interactions: Object
  ▾ interactions: Array (2)
    ▾ 0: Object
        drug: "Pramipexole"
      ▾ brand: Array (3)
          0: "Pramipex"
          1: "Pramipar"
          2: "Prami"
        effect: "SERIOUS"
    ▾ 1: Object
        drug: "Bupropion"
      ▾ brand: Array (3)
          0: "Bupro"
          1: "Bupep"
          2: "Bunocare"
        effect: "MODERATE"
```

4.0.7. Query_7

Title: Filtered medicines by interaction type with a certain substance

Description: This query aims to retrieve information about medications that have interactions involving 'Chlorpromazine' with a defined effect as 'LIFE-THREATENING'. It retrieves the product names and details about drug interactions from up to two matching documents. In particular, the steps are:

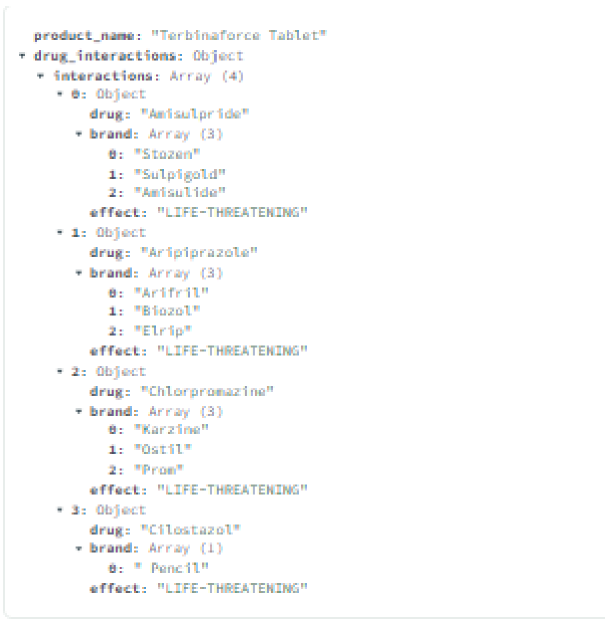
- Conditions: using the first parameter of the find() method
  - Element Matching: Uses the $elemMatch operator to find documents where there is an element (interaction) within the 'drug_interactions.interactions' array that matches both conditions:'drug' equals 'Chlorpromazine' and 'effect' equals 'LIFE-THREATENING'
- Projection: using the second parameter of the find() method, including

  - ’product_name
  - ’drug_interactions’
  - ’_id’ explicitly excluded
- Limiting: just two for readability purposes

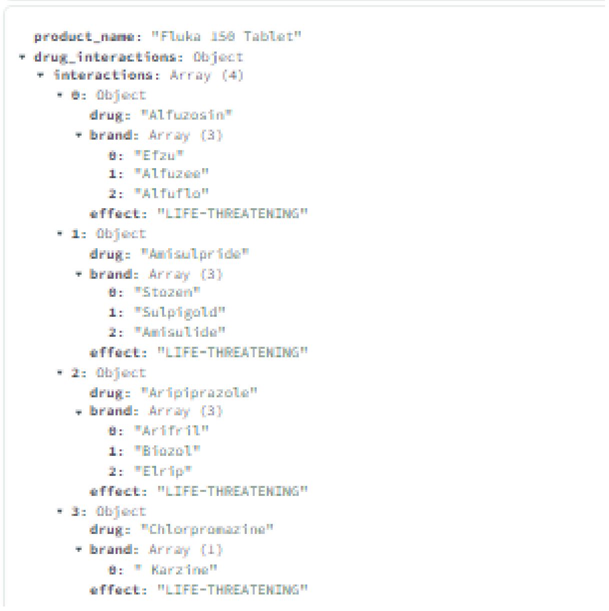
Query:

```
db.getCollection(’medicine_data’)
  .find(
    {
      ’drug_interactions.interactions’: {
        $elemMatch: {
          drug: ’Chlorpromazine’,
          effect: ’LIFE-THREATENING’
        }
      }
    },
    {
      product_name: 1,
      drug_interactions: 1,
      _id: 0
    }
  )
  .limit(2);
```

Query outcome:



```
  product_name: "Terbinaforce Tablet"
▾ drug_interactions: Object
  ▾ interactions: Array (4)
    ▾ 0: Object
        drug: "Amisulpride"
      ▾ brand: Array (3)
          0: "Stozen"
          1: "Sulpigold"
          2: "Amisulide"
        effect: "LIFE-THREATENING"
    ▾ 1: Object
        drug: "Aripiprazole"
      ▾ brand: Array (3)
          0: "Arifril"
          1: "Biozol"
          2: "Elrip"
        effect: "LIFE-THREATENING"
    ▾ 2: Object
        drug: "Chlorpromazine"
      ▾ brand: Array (3)
          0: "Karzine"
          1: "Ostil"
          2: "Prom"
        effect: "LIFE-THREATENING"
    ▾ 3: Object
        drug: "Cilostazol"
      ▾ brand: Array (1)
          0: " Pencil"
        effect: "LIFE-THREATENING"
```



```
  product_name: "Fluka 150 Tablet"
▾ drug_interactions: Object
  ▾ interactions: Array (4)
    ▾ 0: Object
        drug: "Alfuzosin"
      ▾ brand: Array (3)
          0: "Efzu"
          1: "Alfuzee"
          2: "Alfuflo"
        effect: "LIFE-THREATENING"
    ▾ 1: Object
        drug: "Amisulpride"
      ▾ brand: Array (3)
          0: "Stozen"
          1: "Sulpigold"
          2: "Amisulide"
        effect: "LIFE-THREATENING"
    ▾ 2: Object
        drug: "Aripiprazole"
      ▾ brand: Array (3)
          0: "Arifril"
          1: "Biozol"
          2: "Elrip"
        effect: "LIFE-THREATENING"
    ▾ 3: Object
        drug: "Chlorpromazine"
      ▾ brand: Array (1)
          0: " Karzine"
        effect: "LIFE-THREATENING"
```

4.0.8. Query_8

Title: Medications interacting with two specific substances

Description: This query aims to find medications that have interactions with 'Benazepril' while specifically having an interaction element related to the brand 'Enatol' and the drug 'Enalapril'. It retrieves the product names and details about the drug interactions for one matching document. In particular, the steps are:

- Conditions: using the first parameter of the find() method

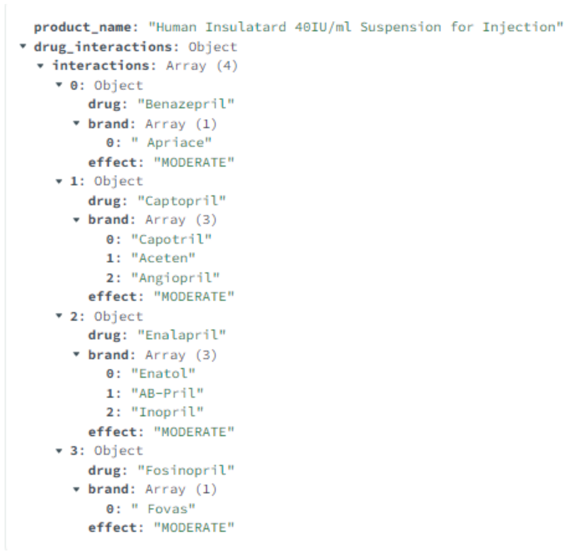
  - First drug: 'drug_interactions.interactions.drug' matches 'Benazepril'.
  - Second drug: 'drug_interactions.interactions' contains an element that matches both the brand 'Enatol' and the drug 'Enalapril'.

- Projection: using the second parameter of the find() method, including
  - 'product_name
  - 'drug_interactions'
  - '_id' explicitly excluded
- Limiting: to a single document for readability purposes

Query:

```
db.getCollection('medicine_data')
  .find(
    {
      'drug_interactions.interactions.drug':
        'Benazepril',
      'drug_interactions.interactions': {
        $elemMatch: {
          brand: 'Enatol',
          drug: 'Enalapril'
        }
      }
    },
    {
      product_name: 1,
      drug_interactions: 1,
      _id: 0
    }
  )
  .limit(1);
```

Query outcome:



```
  product_name: "Human Insulatard 40IU/ml Suspension for Injection"
▾ drug_interactions: Object
  ▾ interactions: Array (4)
    ▾ 0: Object
        drug: "Benazepril"
      ▾ brand: Array (1)
          0: " Apriace"
        effect: "MODERATE"
    ▾ 1: Object
        drug: "Captopril"
      ▾ brand: Array (3)
          0: "Capotril"
          1: "Aceten"
          2: "Angiopril"
        effect: "MODERATE"
    ▾ 2: Object
        drug: "Enalapril"
      ▾ brand: Array (3)
          0: "Enatol"
          1: "AB-Pril"
          2: "Inopril"
        effect: "MODERATE"
    ▾ 3: Object
        drug: "Fosinopril"
      ▾ brand: Array (1)
          0: " Fovas"
        effect: "MODERATE"
```

4.0.9. Query_9

Title: Affordable manufacturers of a type of medication

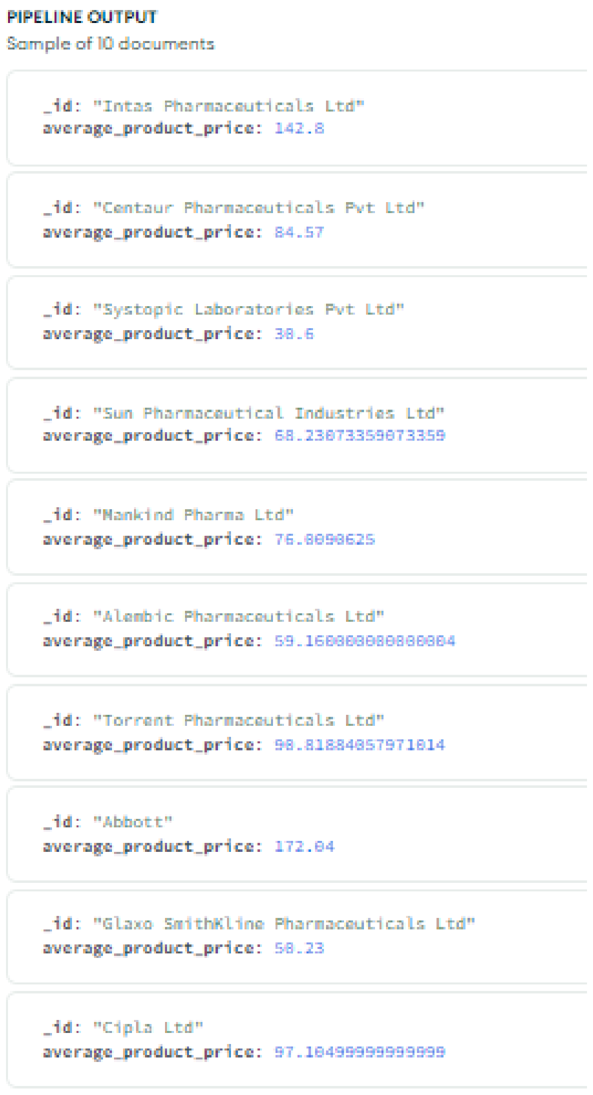
Description: This query identifies and presents 10 manufacturers within the 'Ophthalmological Anti Infectives Medicines' category whose average product price falls at or below 200. In particular, the steps are:

- Match: Filters documents from the 'medicine_data' collection where the 'sub_category' field matches 'Ophthalmological Anti Infectives Medicines'
- Group: Groups the filtered documents by the 'product_manufactured' field, computing the average product price for each manufacturer using the $avg aggregation operator on the 'product_price' field
- Match: Filters the grouped results to include only those manufacturers whose average product price is less than or equal to 200.
- Limit: Restricts the output to the top 10 results for readability purposes

Query:

```
db.getCollection('medicine_data').aggregate(
  [
    {
      $match: {
        sub_category:
          'Ophthalmological Anti Infectives Medicines'
      }
    },
    {
      $group: {
        _id: '$product_manufactured',
        average_product_price: {
          $avg: '$product_price'
        }
      }
    },
    {
      $match: {
        average_product_price: { $lte: 200 }
      }
    },
    { $limit: 10 }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Query outcome:



PIPELINE OUTPUT
Sample of 10 documents

```
_id: "Intas Pharmaceuticals Ltd"
average_product_price: 142.8

_id: "Centaur Pharmaceuticals Pvt Ltd"
average_product_price: 84.57

_id: "Systopic Laboratories Pvt Ltd"
average_product_price: 38.6

_id: "Sun Pharmaceutical Industries Ltd"
average_product_price: 68.23073359073359

_id: "Mankind Pharma Ltd"
average_product_price: 76.8090625

_id: "Alembic Pharmaceuticals Ltd"
average_product_price: 59.16000000000004

_id: "Torrent Pharmaceuticals Ltd"
average_product_price: 98.81884057971014

_id: "Abbott"
average_product_price: 172.04

_id: "Glaxo SmithKline Pharmaceuticals Ltd"
average_product_price: 58.23

_id: "Cipla Ltd"
average_product_price: 97.10499999999999
```

4.0.10. Query_10

Title: Medications count by a certain manufacturer

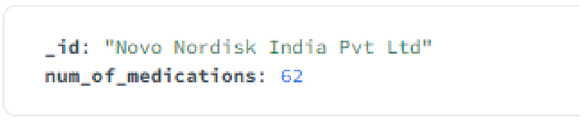
Description: This query aims at determining the total count of medications attributed to the manufacturer 'Novo Nordisk India Pvt Ltd'. In particular, the steps are:

- Match: Filters documents from the 'medicine_data' collection where the 'product_manufactured' field matches 'Novo Nordisk India Pvt Ltd'
- Group: Counts the number of medications manufactured by this company using the $sum aggregation operator

Query:

```
db.getCollection('medicine_data').aggregate(
  [
    {
      $match: {
        product_manufactured:
          'Novo Nordisk India Pvt Ltd'
      }
    },
    {
      $group: {
        _id: 'Novo Nordisk India Pvt Ltd',
        num_of_medications: { $sum: 1 }
      }
    }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Query outcome:



```
_id: "Novo Nordisk India Pvt Ltd"
num_of_medications: 62
```

4.0.11. Query_11

Title: Top 10 manufacturers producing the most expensive medications

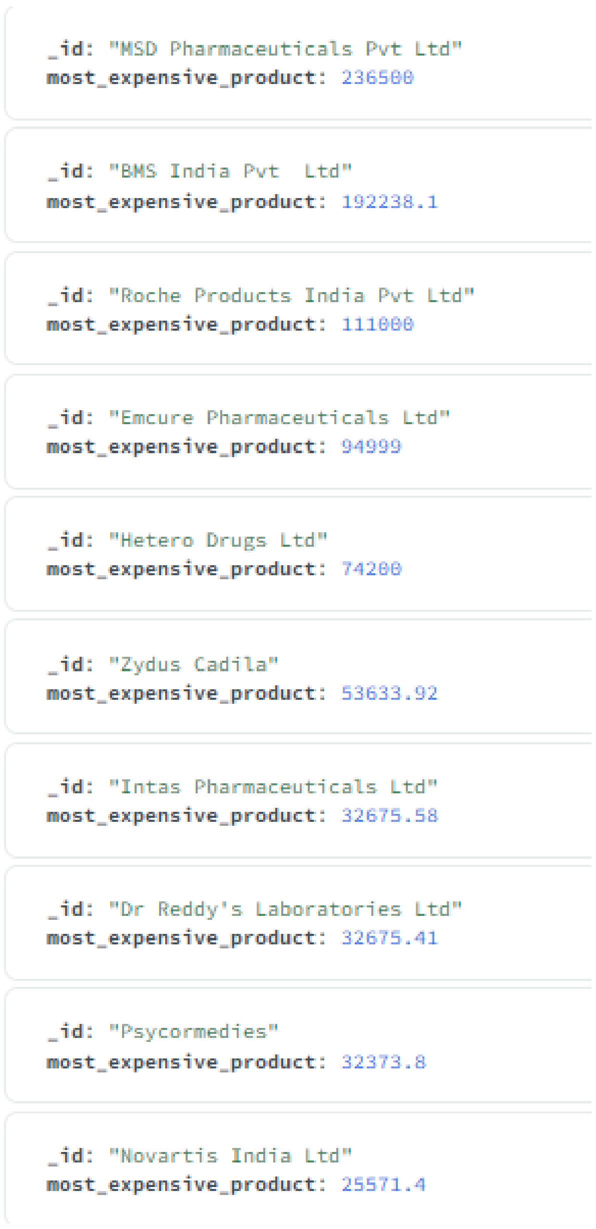
Description: This query aims to provide insights into manufacturers who produce medications with the highest prices. It groups medications by their respective manufacturers, calculates the maximum price for each, and then sorts these manufacturers based on the highest prices, presenting the top 10 manufacturers producing the most expensive medications. In particular, the steps are:

- Group: Groups the documents from the 'medicine_data' collection by the 'product_manufactured' field and determines the maximum product price for each manufacturer using the $max aggregation operator on the 'product_price' field. This determines the most expensive product for each manufacturer
- Sort: Sorts the grouped results in descending order based on the 'most_expensive_product' field, arranging manufacturers based on the highest price of their products

Query:

```
db.getCollection('medicine_data').aggregate(
  [
    {
      $group: {
        _id: '$product_manufactured',
        most_expensive_product: {
          $max: '$product_price'
        }
      }
    },
    { $sort: { most_expensive_product: -1 } },
    { $limit: 10 }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Query outcome:



```
_id: "MSD Pharmaceuticals Pvt Ltd"
most_expensive_product: 236500

_id: "BMS India Pvt  Ltd"
most_expensive_product: 192238.1

_id: "Roche Products India Pvt Ltd"
most_expensive_product: 111000

_id: "Emcure Pharmaceuticals Ltd"
most_expensive_product: 94999

_id: "Hetero Drugs Ltd"
most_expensive_product: 74200

_id: "Zydus Cadila"
most_expensive_product: 53633.92

_id: "Intas Pharmaceuticals Ltd"
most_expensive_product: 32675.58

_id: "Dr Reddy's Laboratories Ltd"
most_expensive_product: 32675.41

_id: "Psycormedies"
most_expensive_product: 32373.8

_id: "Novartis India Ltd"
most_expensive_product: 25571.4
```

4.0.12. Query_12

Title: Less frequent drug interaction brands in a specific sub-category medicine

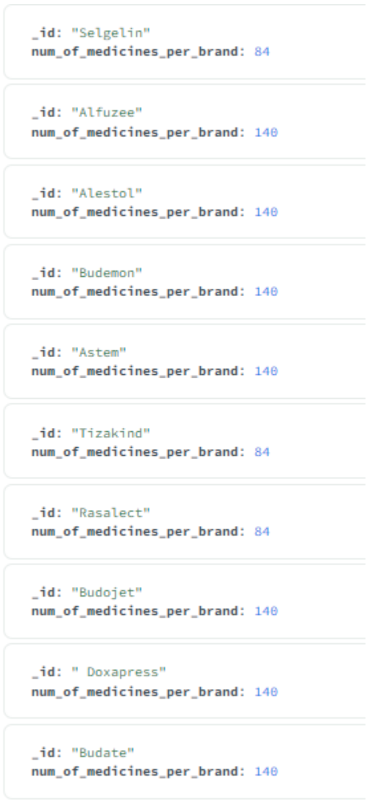
Description: This query aims to identify drug interaction brands within the 'Eye Ear Anti Infectives' sub-category that have fewer than 150 occurrences across medicines. It begins by filtering documents related to the specified sub-category, then unwinds arrays associated with drug interactions and brands, counts medicines per brand, and finally filters to identify less frequent drug interaction brands. In particular, the steps are:

- Match: Filters documents where the 'sub_category' field matches 'Eye Ear Anti Infectives'
- Unwind: Deconstructs the 'drug_interactions.interactions' array and 'drug_interactions.interactions.brand' array, creating new documents for each element within these arrays.
- Group: Groups the documents by each unique 'drug_interactions.interactions.brand' and counts the number of medicines associated with each brand using the $sum aggregation operator
- Match: Filters the grouped results to include only brands with a count of medicines less than 150
- Limit: Limits up to 10 documents for readability purposes

Query:

```
db.getCollection('medicine_data').aggregate(
  [
    {
      $match: {
        sub_category: 'Eye Ear Anti Infectives'
      }
    },
    {
      $unwind: {
        path: '$drug_interactions.interactions'
      }
    },
    {
      $unwind: {
        path: '$drug_interactions.interactions.brand'
      }
    },
    {
      $group: {
        _id: '$drug_interactions.interactions.brand',
        num_of_medicines_per_brand: { $sum: 1 }
      }
    },
    {
      $match: {
        num_of_medicines_per_brand: { $lt: 150 }
      }
    },
    { $limit: 10 }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Query outcome:



```
_id: "Selgelin"
num_of_medicines_per_brand: 84

_id: "Alfuzee"
num_of_medicines_per_brand: 140

_id: "Alestol"
num_of_medicines_per_brand: 140

_id: "Budemon"
num_of_medicines_per_brand: 140

_id: "Astem"
num_of_medicines_per_brand: 140

_id: "Tizakind"
num_of_medicines_per_brand: 84

_id: "Rasalect"
num_of_medicines_per_brand: 84

_id: "Budojet"
num_of_medicines_per_brand: 140

_id: " Doxapress"
num_of_medicines_per_brand: 140

_id: "Budate"
num_of_medicines_per_brand: 140
```

4.0.13. Query_13

Title: Top 10 manufacturers per category with the highest product counts

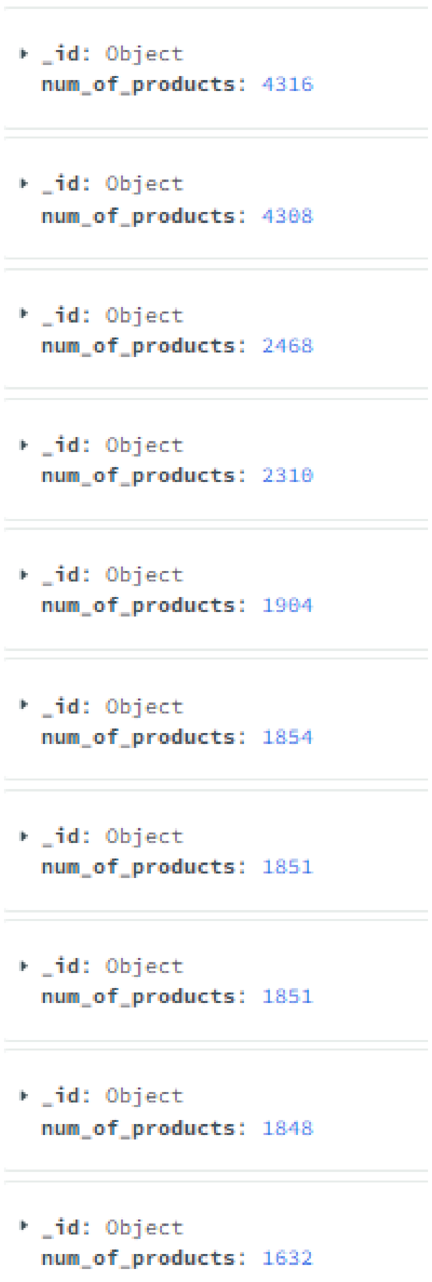
Description: This query aims to identify and present the top 10 combinations of product categories and manufacturers that have the highest counts of associated products. It groups documents by both the sub-category and manufacturer, calculates the number of products for each combination, and then sorts these combinations based on the highest product counts. In particular, the steps are:

- Group: Groups the documents by a composite identifier consisting of 'sub_category' and 'product_manufactured'. It counts the number of products per combination using the $sum aggregation operator
- Sort: Sorts the grouped results in descending order based on the count of products ('num_of_products'), ensuring combinations with the highest product counts appear at the top.
- Limit: Limits the output to the top 10 combinations of 'sub_category' and 'product_manufactured' with the highest product counts.

Query:

```
db.getCollection('medicine_data').aggregate(
  [
    {
      $group: {
        _id: {
          sub_categrory: '$sub_category',
          product_manufactured:
            '$product_manufactured'
        },
        num_of_products: { $sum: 1 }
      }
    },
    { $sort: { num_of_products: -1 } },
    { $limit: 10 }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Query outcome:


```
▸ _id: Object
  num_of_products: 4316

▸ _id: Object
  num_of_products: 4308

▸ _id: Object
  num_of_products: 2468

▸ _id: Object
  num_of_products: 2310

▸ _id: Object
  num_of_products: 1904

▸ _id: Object
  num_of_products: 1854

▸ _id: Object
  num_of_products: 1851

▸ _id: Object
  num_of_products: 1851

▸ _id: Object
  num_of_products: 1848

▸ _id: Object
  num_of_products: 1632
```

4.0.14. Query_14

Title: Manufacturers of Vitamin K Antagonists with low average price

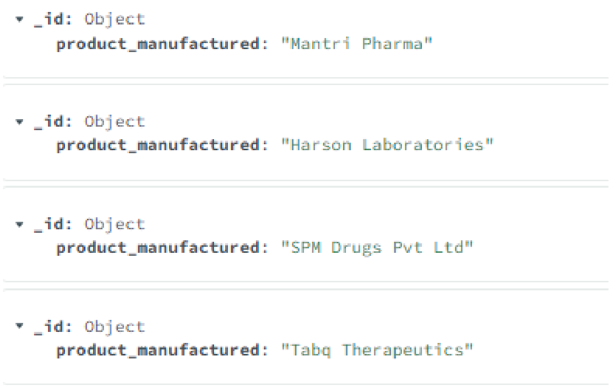
Description: This query aims to identify manufacturers producing 'Vitamin K Antagonists' whose average price across their products falls below 10. It groups documents by both the 'Vitamin K Antagonists' category and manufacturer, calculates the total price for each manufacturer, and filters for those with an average price below the specified threshold. In particular, the steps are:

- Group: Groups the documents by a composite identifier consisting of 'sub_category' (renamed as 'category') and 'product_manufactured'. It calculates the total price for each manufacturer using the $sum aggregation operator on the 'product_price' field
- Match: Filters the results to include only documents where the '_id.category' is 'Vitamin K Antagonists'
- Match: Further filters the results to consider only those with an 'avg_price' (total price) less than 10
- Project: Projects only the 'product_manufactured' field in the output, renaming the identifier '_id.product_manufactured' as it represents the manufacturer

Query:

```
db.getCollection('medicine_data').aggregate(
  [
    {
      $group: {
        _id: {
          category: '$sub_category',
          product_manufactured:
            '$product_manufactured'
        },
        avg_price: { $sum: '$product_price' }
      }
    },
    {
      $match: {
        '_id.category': 'Vitamin K Antagonists'
      }
    },
    { $match: { avg_price: { $lt: 10 } } },
    {
      $project: { '_id.product_manufactured': 1 }
    }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Query outcome:



```
▾ _id: Object
    product_manufactured: "Mantri Pharma"

▾ _id: Object
    product_manufactured: "Harson Laboratories"

▾ _id: Object
    product_manufactured: "SPM Drugs Pvt Ltd"

▾ _id: Object
    product_manufactured: "Tabq Therapeutics"
```

4.0.15. Query_15

Title: Drug Interactions with specific components and effect

Description: This query aims to identify and retrieve drug interaction information involving 'Benazepril' with a 'SERIOUS' effect or 'Enalapril' with a 'MODERATE' effect It begins by unwinding the drug interactions, filters for interactions meeting the specified drug and effect conditions, and finally projects the relevant drug interaction details. In particular, the steps are:

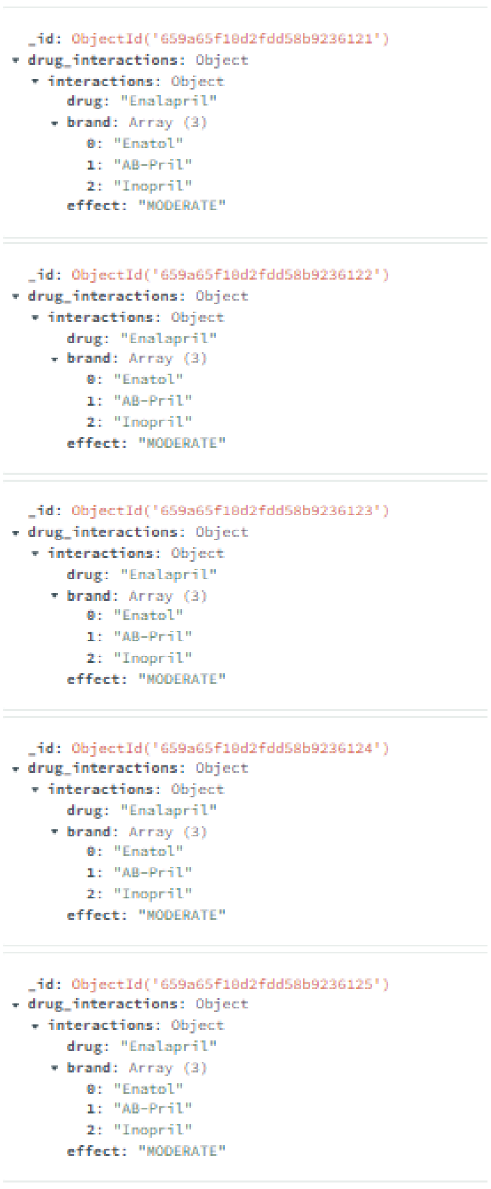
- Unwind: Deconstructs the 'drug_interactions.interactions' array, creating a new document for each element within the array
- Match: Filters documents to include only those where the drug interactions meet specific conditions: either 'Benazepril' with a 'SERIOUS' effect or 'Enalapril' with a 'MODERATE' effect
- Project: Projects only the 'drug_interactions' field in the output, retaining information related to the drug interactions

- Limit: 10 for readability purposes

Query:

```
db.getCollection('medicine_data').aggregate(
  [
    {
      $unwind: {
        path: '$drug_interactions.interactions'
      }
    },
    {
      $match: {
        $or: [
          {
            'drug_interactions.interactions.drug':
              'Benazepril',
            'drug_interactions.interactions.effect':
              'SERIOUS'
          },
          {
            'drug_interactions.interactions.drug':
              'Enalapril',
            'drug_interactions.interactions.effect':
              'MODERATE'
          }
        ]
      }
    },
    { $project: { drug_interactions: 1 } },
    { $limit: 10 }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Query outcome:



```
_id: ObjectId('659a65f10d2fdd58b9236121')
▾ drug_interactions: Object
  ▾ interactions: Object
      drug: "Enalapril"
    ▾ brand: Array (3)
        0: "Enatol"
        1: "AB-Pril"
        2: "Inopril"
      effect: "MODERATE"

_id: ObjectId('659a65f10d2fdd58b9236122')
▾ drug_interactions: Object
  ▾ interactions: Object
      drug: "Enalapril"
    ▾ brand: Array (3)
        0: "Enatol"
        1: "AB-Pril"
        2: "Inopril"
      effect: "MODERATE"

_id: ObjectId('659a65f10d2fdd58b9236123')
▾ drug_interactions: Object
  ▾ interactions: Object
      drug: "Enalapril"
    ▾ brand: Array (3)
        0: "Enatol"
        1: "AB-Pril"
        2: "Inopril"
      effect: "MODERATE"

_id: ObjectId('659a65f10d2fdd58b9236124')
▾ drug_interactions: Object
  ▾ interactions: Object
      drug: "Enalapril"
    ▾ brand: Array (3)
        0: "Enatol"
        1: "AB-Pril"
        2: "Inopril"
      effect: "MODERATE"

_id: ObjectId('659a65f10d2fdd58b9236125')
▾ drug_interactions: Object
  ▾ interactions: Object
      drug: "Enalapril"
    ▾ brand: Array (3)
        0: "Enatol"
        1: "AB-Pril"
        2: "Inopril"
      effect: "MODERATE"
```

4.0.16. Query_16

Title: Average price per drug interaction brand

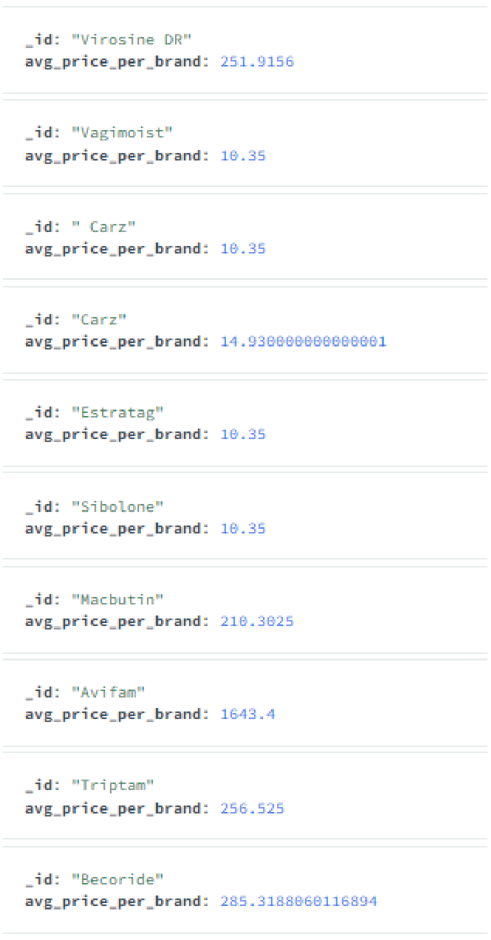
Description: This query aims to compute and present the average price per drug interaction brand. It starts by filtering documents with existing product prices, then unwinds arrays related to drug interactions and their associated brands, calculates the average price for each brand, and finally limits the output to the top 10 brands based on their average prices. In particular, the steps are:

- Match: Filters documents where 'product_price' field exists
- Unwind: Deconstructs the 'drug_interactions.interactions' array and 'drug_interactions.interactions.brand' array, creating new documents for each element within these arrays
- Group: Groups the documents by each unique 'drug_interactions.interactions.brand' and calculates the average product price for each brand using the $avg aggregation operator
- Limit: Limits the output to the top 10 drug interaction brands based on their average price

Query:

```
db.getCollection('medicine_data').aggregate(
  [
    {
      $match: { product_price: { $exists: true } }
    },
    {
      $unwind: {
        path: '$drug_interactions.interactions'
      }
    },
    {
      $unwind: {
        path: '$drug_interactions.interactions.brand'
      }
    },
    {
      $group: {
        _id: '$drug_interactions.interactions.brand',
        avg_price_per_brand: {
          $avg: '$product_price'
        }
      }
    },
    { $limit: 10 }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Query outcome:



```
_id: "Virosine DR"
avg_price_per_brand: 251.9156

_id: "Vagimoist"
avg_price_per_brand: 10.35

_id: " Carz"
avg_price_per_brand: 10.35

_id: "Carz"
avg_price_per_brand: 14.930000000000001

_id: "Estratag"
avg_price_per_brand: 10.35

_id: "Sibolone"
avg_price_per_brand: 10.35

_id: "Macbutin"
avg_price_per_brand: 210.3025

_id: "Avifam"
avg_price_per_brand: 1643.4

_id: "Triptam"
avg_price_per_brand: 256.525

_id: "Becoride"
avg_price_per_brand: 285.3188060116894
```

4.0.17. Query_17

Title: Top 10 side effects and their occurrences

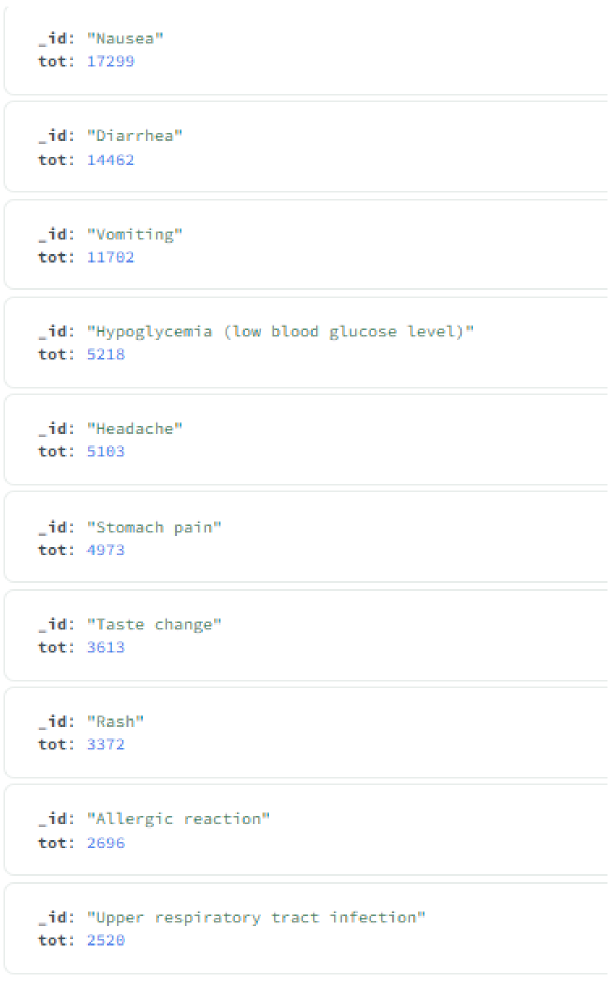
Description: This query aims to retrieve information about medications that have a specific salt composition dose ('500mg/5ml') and precisely four side effects listed in their data. It extracts the product name and medicine description from the matching documents. In particular, the steps are:

- Unwind: Deconstructs the 'side_effects_type' array, creating a new document for each element within the array
- Group: Groups the documents by each unique 'side_effects_type', counting the occurrences of each side effect using the $sum aggregation operator
- Sort: Sorts the grouped results in descending order based on the count of occurrences ('tot'), ensuring the side effects with the highest frequencies appear at the top
- Limit: Limits the output to the top 10 side effects with the highest occurrences

Query:

```
db.getCollection('medicine_data').aggregate(
  [
    { $unwind: { path: '$side_effects_type' } },
    {
      $group: {
        _id: '$side_effects_type',
        tot: { $sum: 1 }
      }
    },
    { $sort: { tot: -1 } },
    { $limit: 10 }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Query outcome:



```
_id: "Nausea"
tot: 17299

_id: "Diarrhea"
tot: 14462

_id: "Vomiting"
tot: 11702

_id: "Hypoglycemia (low blood glucose level)"
tot: 5218

_id: "Headache"
tot: 5103

_id: "Stomach pain"
tot: 4973

_id: "Taste change"
tot: 3613

_id: "Rash"
tot: 3372

_id: "Allergic reaction"
tot: 2696

_id: "Upper respiratory tract infection"
tot: 2520
```

4.0.18. Query_18

Title: Minimum average price among salt compositions

Description: This query aims to find and present the minimum average price among different salt compositions. It starts by unwinding the compositions, calculates the average price for each composition, and then determines the minimum average price across all these compositions. In particular, the steps are:

- Conditions: using the first parameter of the find() method
  - Logical AND Operator: The $and operator is utilized to specify that both conditions must be satisfied.
  - Salt Composition Dose: The query filters documents where the 'salt_composition.dose' field matches '500mg/5ml'

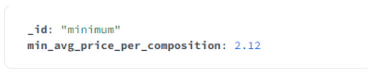
    - Side Effects Type: It further filters documents where the 'side_effects_type' array field has precisely 4 elements

- Projection: using the second parameter of the find() method, including
    - 'product_name
    - 'medicine_desc'

Query:

```
db.getCollection('medicine_data').aggregate(
  [
    {
      $unwind: {
        path: '$salt_composition.composition'
      }
    },
    {
      $group: {
        _id: '$salt_composition.composition',
        avg_price_per_composition: {
          $avg: '$product_price'
        }
      }
    },
    {
      $group: {
        _id: 'minimum',
        min_avg_price_per_composition: {
          $min: '$avg_price_per_composition'
        }
      }
    }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Query outcome:



```
_id: "minimum"
min_avg_price_per_composition: 2.12
```

4.0.19. Query_19

Title: Top 10 product-composition-side effect combinations

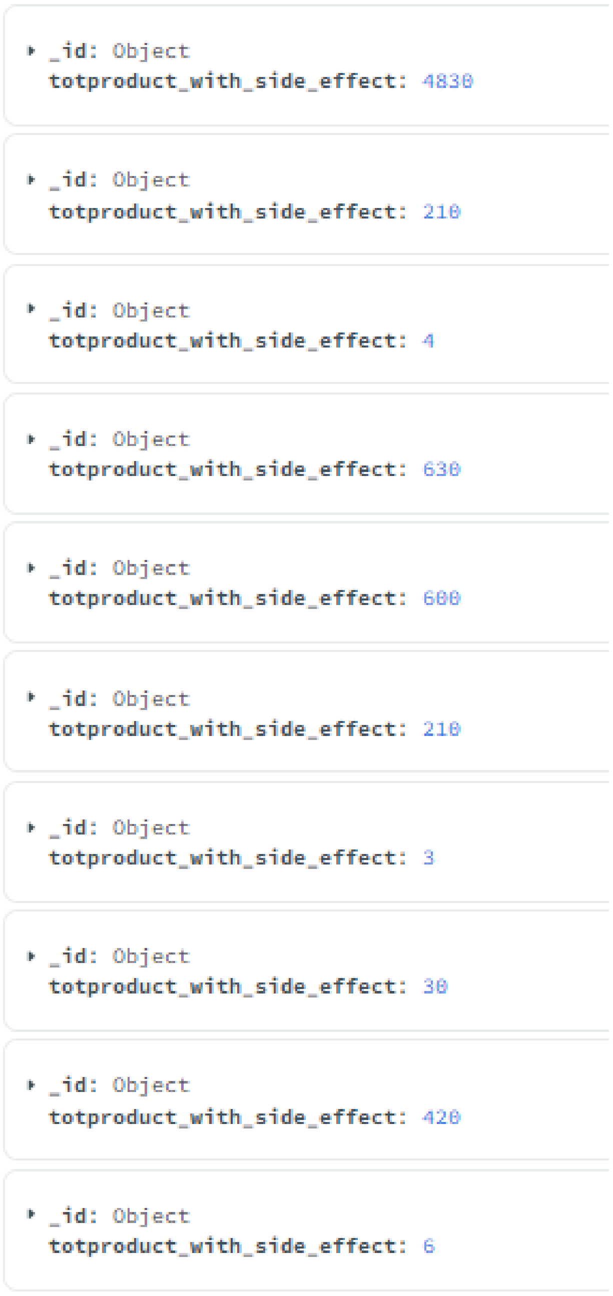
Description: This query aims to identify and present the top 10 combinations of product compositions and side effects with the highest occurrences. It unwinds both the composition and side effects arrays, groups documents based on these combinations, calculates the total number of products associated with each combination, and finally limits the output to the top 10 combinations. In particular, the steps are:

- Unwind: Deconstructs the 'salt_composition.composition' array and 'side_effects_type' array, creating new documents for each element within these arrays
- Group: Groups the documents based on combinations of 'salt_composition.composition' and 'side_effects_type'. It calculates the total number of products associated with each composition-side effect combination using the $sum aggregation operator
- Limits the output to the top 10 combinations of product compositions and side effects with the highest occurrence counts

Query:

```
db.getCollection('medicine_data').aggregate(
  [
    {
      $unwind: {
        path: '$salt_composition.composition'
      }
    },
    { $unwind: { path: '$side_effects_type' } },
    {
      $group: {
        _id: {
          composition:
            '$salt_composition.composition',
          side_effects_type: '$side_effects_type'
        },
        totproduct_with_side_effect: { $sum: 1 }
      }
    },
    { $limit: 10 }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Query outcome:



```
▸ _id: Object
  totproduct_with_side_effect: 4830

▸ _id: Object
  totproduct_with_side_effect: 210

▸ _id: Object
  totproduct_with_side_effect: 4

▸ _id: Object
  totproduct_with_side_effect: 630

▸ _id: Object
  totproduct_with_side_effect: 600

▸ _id: Object
  totproduct_with_side_effect: 210

▸ _id: Object
  totproduct_with_side_effect: 3

▸ _id: Object
  totproduct_with_side_effect: 30

▸ _id: Object
  totproduct_with_side_effect: 420

▸ _id: Object
  totproduct_with_side_effect: 6
```

4.0.20. Query_20

Title: Top 10 drug compositions with most duplicated side effects

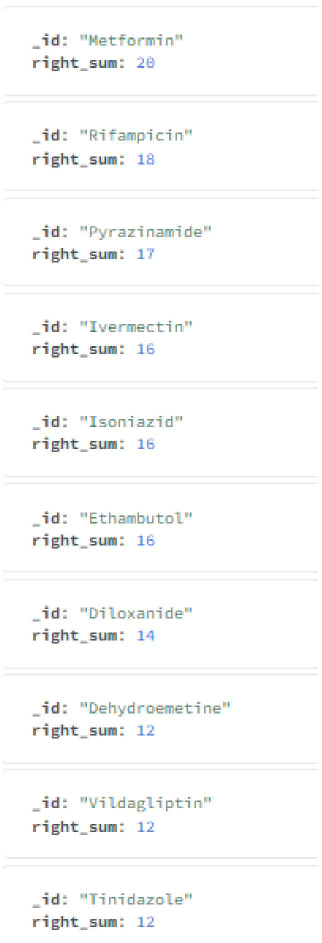
Description: This query aims to identify and present the top 10 drug compositions with the most occurrences of unique duplicated side effects. It performs the unwinding of arrays, groups documents by drug composition and side effects, calculates counts, and then sorts and limits the output to the top 10 compositions. In particular, the steps are:

- Unwind: Deconstructs the 'side_effects_type' array and 'salt_composition.composition' array, creating new documents for each element within these arrays.
- Group: Groups the documents based on combinations of 'salt_composition.composition' and 'side_effects_type'. It calculates the count of duplicated side effects associated with each drug composition
- Groups the results by the drug compositions and calculates the total count of unique duplicated side effects for each drug composition
- Sort: Sorts the grouped results in descending order based on the count of unique duplicated side effects ('right_sum'), ensuring the compositions with the most duplicated side effects are at the top
- Limit: Limits the output to the top 10 drug compositions with the highest counts of unique duplicated side effects

Query:

```
db.getCollection('medicine_data').aggregate(
  [
    { $unwind: { path: '$side_effects_type' } },
    {
      $unwind: {
        path: '$salt_composition.composition'
      }
    },
    {
      $group: {
        _id: {
          drug_composition:
            '$salt_composition.composition',
          duplicate_side_effect:
            '$side_effects_type'
        },
        count_dup: { $sum: 1 }
      }
    },
    {
      $group: {
        _id: '$_id.drug_composition',
        right_sum: { $sum: 1 }
      }
    },
    { $sort: { right_sum: -1 } },
    { $limit: 10 }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);
```

Query outcome:



```
_id: "Metformin"
right_sum: 20

_id: "Rifampicin"
right_sum: 18

_id: "Pyrazinamide"
right_sum: 17

_id: "Ivermectin"
right_sum: 16

_id: "Isoniazid"
right_sum: 16

_id: "Ethambutol"
right_sum: 16

_id: "Diloxanide"
right_sum: 14

_id: "Dehydroemetine"
right_sum: 12

_id: "Vildagliptin"
right_sum: 12

_id: "Tinidazole"
right_sum: 12
```