Assignment 5

Hunjun Shin

MongoDB

1. Load the file restaurants.json into a MongoDB database. The file contains data on 3,772 restaurants in New York City.

Write the following queries in MongoDB (show the output of your queries):

a) Display all the restaurants located in the boroughs Bronx or Brooklyn.

```
// 1.a

db.res.find(
    {borough: {$in: ['Bronx', 'Brooklyn']}},
    {_id:0, name:1, borough:1}
);
```


```
[
  {
    "borough": "Brooklyn",
    "name": "Wilken'S Fine Food"
  },
  {
    "borough": "Bronx",
    "name": "Wild Asia"
  },
  {
    "borough": "Brooklyn",
    "name": "Taste The Tropics Ice Cream"
  },
  {
    "borough": "Brooklyn",
    "name": "Wendy'S"
  },
  {
    "borough": "Brooklyn",
    "name": "C & C Catering Service"
  },
  {
    "borough": "Brooklyn",
    "name": "May May Kitchen"
  },
  {
    "borough": "Bronx",
    "name": "Morris Park Bake Shop"
  },
  {
    "borough": "Brooklyn",
    "name": "Riviera Caterer"
  },
  {
    "borough": "Brooklyn",
    "name": "Regina Caterers"
  },
```


b) Find the restaurant id, name, borough and cuisine for those restaurants whose name starts with the letters 'Mad'.

```
db.res.find(
    {name: /^Mad/}, {restaurant_id:1, name :1, borough:1, cuisine:1}
);
```


```
{
  "_id": {
    "$oid": "6730f2812a15430da03bb897"
  },
  "borough": "Manhattan",
  "cuisine": "American ",
  "name": "Madison Square",
  "restaurant_id": "40402527"
},
{
  "_id": {
    "$oid": "6730f2812a15430da03bb96a"
  },
  "borough": "Manhattan",
  "cuisine": "Indian",
  "name": "Madras Mahal",
  "restaurant_id": "40519978"
},
{
  "_id": {
    "$oid": "6730f2812a15430da03bbc1c"
  },
  "borough": "Manhattan",
  "cuisine": "American ",
  "name": "Madame X",
  "restaurant_id": "40611024"
},
{
  "_id": {
    "$oid": "6730f2812a15430da03bbcce"
  },
  "borough": "Manhattan",
  "cuisine": "French",
  "name": "Madison Bistro",
  "restaurant_id": "40657588"
},
```


3. Find the restaurants that have received a score between 80 and 90 (inclusive).

```
// 3.a

db.res.find(
    {grades:
        {$elemMatch:
            {score: {$gte: 80, $lte:90}}
        }},
    {name:1, _id:0}
);
```


```
[
  {
    "name": "B.B. Kings"
  },
  {
    "name": "West 79Th Street Boat Basin Cafe"
  }
]
```

4. Display the restaurant id and name of restaurants which have received a 'C' grade in year 2014.

```
db.res.find(
    {grades:
        {$elemMatch:
            {grade: 'C', date:
                {$gte: ISODate('2014-01-01'), $lt: ISODate('2015-01-01')}
            }
        }
    },
    {restaurant_id:1, name:1, _id:0}
);
```


```
[
  {
    "name": "B & M Hot Bagel & Grocery",
    "restaurant_id": "40364299"
  },
  {
    "name": "Texas Rotisserie",
    "restaurant_id": "40364304"
  },
  {
    "name": "Nyac Main Dining Room",
    "restaurant_id": "40364467"
  },
  {
    "name": "Mitchell'S Restaurant",
    "restaurant_id": "40366961"
  },
  {
    "name": "Mcdonald'S",
    "restaurant_id": "40370781"
  },
  {
    "name": "Burger King",
    "restaurant_id": "40370916"
  },
  {
    "name": "Omonia Cafe",
    "restaurant_id": "40372445"
  },
  {
    "name": "Murals On 54/Randolphs'S",
    "restaurant_id": "40372466"
  },
  {
    "name": "Caffe Dante",
    "restaurant_id": "40373149"
  },
```


5.Find the cuisine that has the highest number of restaurants.

```
db.res.aggregate(
    {$group: {_id: '$cuisine', total: { $count: {}}}},
    {$sort: {total:-1}},
    {$limit: 1}
);
```


```
[
  {
    "_id": "American ",
    "total": 1255
  }
]
```


6. Find the restaurants that do not prepare an 'American' cuisine and their average grade score is higher than 30. Display the restaurant ids and their average score.

```
db.res.aggregate(
    {$unwind: "$grades"},
    {$group: {_id: '$restaurant_id', average_grade: {$avg: "$grades.score"}}},
    {$match: { cuisine: {$ne: 'American'}, average_grade: {$gt: 30}}},
    {$project: {_id:1 , average_grade:1}}
);
```


```
[
  {
    "_id": "40825993",
    "average_grade": 30.8
  },
  {
    "_id": "40624470",
    "average_grade": 30.6
  },
  {
    "_id": "40374268",
    "average_grade": 30.8
  },
  {
    "_id": "40387237",
    "average_grade": 32.6
  },
  {
    "_id": "40366157",
    "average_grade": 32.142857142857146
  },
  {
    "_id": "40393488",
    "average_grade": 38.6
  },
  {
    "_id": "40372466",
    "average_grade": 33.666666666666664
  },
  {
    "_id": "40756344",
    "average_grade": 36
  }
```


7.(g) For each restaurant display only the grades that were recorded from the year 2014 onwards.

```
db.res.aggregate(
    {$project: {
        name: 1,
        grades : {
            $filter: {
                input: "$grades",
                as: "grade",
                cond: {$gte: ["$$grade.date", ISODate("2014-01-01")]}
            }
        }
    }}
);
```


```
[
  {
    "_id": {
      "$oid": "6730f2812a15430da03bb360"
    },
    "name": "Brunos On The Boulevard",
    "grades": [
      {
        "date": {
          "$date": "2014-11-15T00:00:00Z"
        },
        "grade": "Z",
        "score": 38
      },
      {
        "date": {
          "$date": "2014-05-02T00:00:00Z"
        },
        "grade": "A",
        "score": 10
      }
    ]
  },
  {
    "_id": {
      "$oid": "6730f2812a15430da03bb361"
    },
    "name": "Wilken'S Fine Food",
    "grades": [
      {
        "date": {
          "$date": "2014-05-29T00:00:00Z"
        },
        "grade": "A",
        "score": 10
      },
      {
        "date": {
          "$date": "2014-01-14T00:00:00Z"
        },
        "grade": "A",
        "score": 10
      }
    ]
  },
```


8.(h) Calculate the average score across all the restaurants in the collection.

```
//8.(h)  Calculate the average score across all the restaurants in the collection.

db.res.aggregate(
    {$unwind: "$grades"},
    {$group:{_id: null, tot_average_score: {$avg: '$grades.score'}}}
);
```

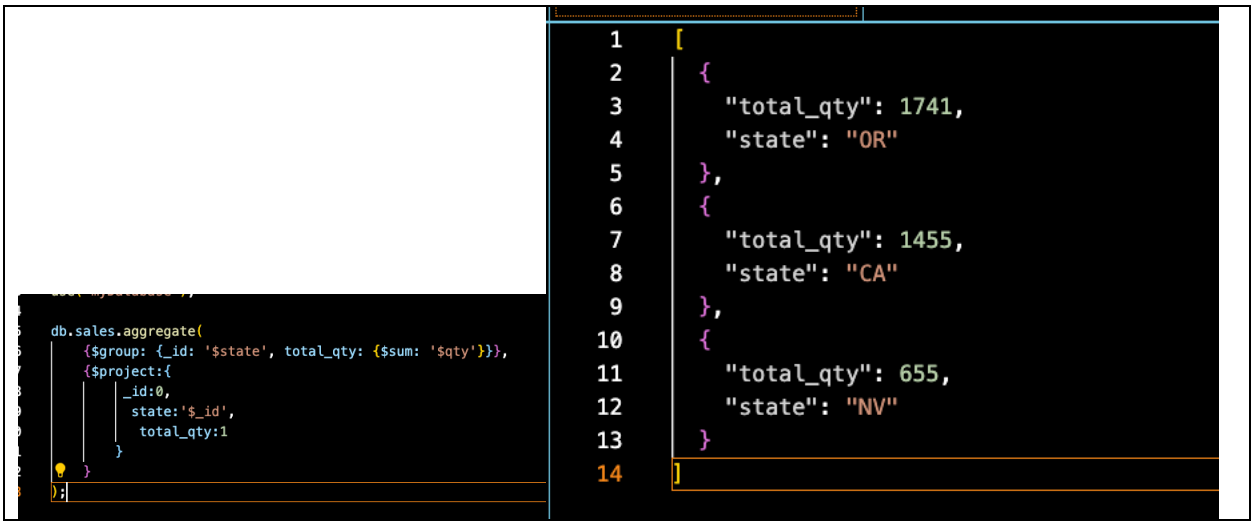
Playground Result


```
[
  {
    "_id": null,
    "tot_average_score": 11.427736743468195
  }
]
```

2. Create a collection named sales with the following documents:

```
{ "_id": 1, "city": "Berkeley", "state": "CA", "qty": 648 },
{ "_id": 2, "city": "Bend", "state": "OR", "qty": 491 },
{ "_id": 3, "city": "Kensington", "state": "CA", "qty": 233 },
{ "_id": 4, "city": "Eugene", "state": "OR", "qty": 842 },
{ "_id": 5, "city": "Reno", "state": "NV", "qty": 655 },
{ "_id": 6, "city": "Portland", "state": "OR", "qty": 408 },
{ "_id": 7, "city": "Sacramento", "state": "CA", "qty": 574 }
```

a)



```
db.sales.aggregate(
    {$group: {_id: '$state', total_qty: {$sum: '$qty'}}},
    {$project:{
        _id:0,
        state:'$_id',
        total_qty:1
    }
  }
);
```

```
[
  {
    "total_qty": 1741,
    "state": "OR"
  },
  {
    "total_qty": 1455,
    "state": "CA"
  },
  {
    "total_qty": 655,
    "state": "NV"
  }
]
```

b)

i.

```
db.sales.updateOne(
    {city: "Berkely"},
    {$set: {qty: 750}}
);
```

ii.

```
db.saels.updateMany(
    {state: 'OR'},
    {$inc: {qty: 50}}
);
```

iii.

```
db.saels.updateOne(
    {_id: 5},
    {$set: {salespeople: ["David", "Martha"]}}
);
```

iv.

```
db.sales.updateOne(
    {_id: 5},
    {$push: {salespeople: "James"}}
);
```

v.

```
db.sales.updateOne(
    {_id:5, salespeople: "Martha"},
    { $set: { "salespeople.$": "Lisa"}}
);
```

vi.

```
db.sales.deleteMany(
    { state: "CA" } // 조건
  );

db.sales.find();
```

c) Print the final sales collection

```
myDatabase> db.sales.find()
[
  { _id: 2, city: 'Bend', state: 'OR', qty: 541 },
  { _id: 4, city: 'Eugene', state: 'OR', qty: 892 },
  {
    _id: 5,
    city: 'Reno',
    state: 'NV',
    qty: 655,
    salespeople: [ 'David', 'Lisa', 'James' ]
  },
  { _id: 6, city: 'Portland', state: 'OR', qty: 458 }
]
```

3. Load the file books.json into a MongoDB database. Write a Python script that lets the user enter a books category and prints the ISBN and titles of all the books in that category.

```python
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")
db = client['myDatabase']  # 데이터베이스 이름
collection = db['books']   # 컬렉션 이름

category = input("Enter category: ")

books = collection.find({"categories": category}, {"isbn": 1, "title": 1, "_id": 0})

found = False
for book in books:
    print(f"ISBN: {book['isbn']}, title: {book['title']}")
    found = True

if not found:
    print("No books found in this category.")
```

```
(base) hunjunsin@hunjunui-MacBookAir mongoDB % /Users/hunjunsin/.pyenv/versions/3.11.
python /Users/hunjunsin/Desktop/box/Introduction_DMB/mongoDB/books.py
Enter category: Computer Graphics
ISBN: 1884777902, title: 3D User Interfaces with Java 3D
ISBN: 133034054, title: Graphics File Formats
ISBN: 1933988398, title: Gnuplot in Action
ISBN: 1884777473, title: The Awesome Power of Direct3D/DirectX
ISBN: 138412146, title: Power-3D
ISBN: 1930110022, title: Graphics Programming with Perl
(base) hunjunsin@hunjunui-MacBookAir mongoDB %
```

MapReduce

1. Projection πS(R): From each tuple of relation R produce only the components for the attributes in S.

1.map(key, value): 2. projected_value = {attribute in S} 3. emit(projected_value, None)
1.reduce(projected_value, _): 2. emit(projected_value, None)

2. Intersection R∩S: Return the tuples that are present in both relations R and S. Assume that relations R and S have the same schema (same attributes and same type).

```
1.map(tuple):
2.    emit(tuple, "R") //  add R tag if tuple comes from R
3.    emit(tuple, "S") //  add S tag if tuple comes from S
```

```
1.  reduce(tuple, tags):
2.      // tags is the list of tag ['R','S']
3.      if "R" in tags and "S" in tags:
4.      emit(tuple, None) // tuple 이 R 과 S 모두에 존재하면 반환
```

3.Grouping γA,θ(B)(R). Given a relation R(A, B, C), with one grouping attribute A, one aggregated attribute B, and another attribute C, which is neither grouped or aggregated:

(a) Partition the tuples of R according to their values in attribute A.
(b) For each group, aggregate the values in attribute B and apply function θ on the

aggregated value (θ is an aggregation operation such as SUM, COUNT or MAX).

The result of this operation is one tuple for each group. That tuple has a component for the grouping attribute A, with the value common to tuples of that group. It also has a component for each aggregation θ(B), with the aggregated value for that group.

```
a)
map(tuple):
  // tuple: (A, B, C)
  key = tuple.A
  value = tuple.B
  emit(key, value)
```

```
b)
reduce(key, value_list):
  aggregated_value = sum(value_list)
  emit(key, aggregated_value)
```

Spark

1. Show the total number of movies in each genre with (a) the DataFrame API, (b) Spark SQL, and (c) RDD operations. Identify the most efficient method.

DataFrame API

```
from pyspark.sql.functions import explode, col

genre_count_df = movies_df.select(explode(col("genres")).alias("genre")).groupBy('genre').count()

genre_count_df.show()
```

```
+-----------+-----+
|      genre|count|
+-----------+-----+
|      Crime| 2678|
|    Romance| 3665|
|   Thriller| 2658|
|  Adventure| 2045|
|      Drama|13789|
|        War|  794|
|Documentary| 2129|
|     Family| 1311|
|    Fantasy| 1153|
|    History|  999|
|    Mystery| 1259|
|    Musical|  487|
|  Animation|  971|
|      Music|  840|
|  Film-Noir|  105|
|      Short|  478|
|     Horror| 1703|
|    Western|  274|
|  Biography| 1404|
|     Comedy| 7024|
+-----------+-----+
only showing top 20 rows
```

Spark SQL

```
# Register DataFrame as a SQL temporary view
movies_df.createOrReplaceTempView("movies")

# Execute SQL query
genre_count_sql = spark.sql("""
    SELECT genre, COUNT(*) as count
    FROM (SELECT EXPLODE(genres) AS genre FROM movies)
    GROUP BY genre
""")
genre_count_sql.show()
```

```
24/11/14 13:34:42 WARN Nat
+-----------+-----+
|      genre|count|
+-----------+-----+
|      Crime| 2678|
|    Romance| 3665|
|   Thriller| 2658|
|  Adventure| 2045|
|      Drama|13789|
|        War|  794|
|Documentary| 2129|
|     Family| 1311|
|    Fantasy| 1153|
|    History|  999|
|    Mystery| 1259|
|    Musical|  487|
|  Animation|  971|
|      Music|  840|
|  Film-Noir|  105|
|      Short|  478|
|     Horror| 1703|
|    Western|  274|
|  Biography| 1404|
|     Comedy| 7024|
+-----------+-----+
only showing top 20 rows
```

RDD

```
genre_count_rdd = movies_df.select("genres").rdd.\
    filter(lambda row: row["genres"] is not None).\
        flatMap(lambda row: row["genres"]).\
        map(lambda genre: (genre, 1)) .reduceByKey(lambda a, b: a + b)

print(genre_count_rdd.collect())
```

```
l psutil to have better support with spilling
[('Romance', 3665), ('Horror', 1703), ('Sci-Fi', 1034), ('News', 51), ('History', 999), ('Adventure', 2045), ('Documentary', 2129), ('Crime', 2678),
('Mystery', 1259), ('Film-Noir', 105), ('Talk-Show', 1), ('Comedy', 7024), ('Sport', 390), ('Western', 274), ('Animation', 971), ('Family', 1311), ('
Thriller', 2658), ('Drama', 13789), ('Action', 2539), ('Music', 840), ('Musical', 487), ('Short', 478), ('Fantasy', 1153), ('Biography', 1404), ('War
', 794)]
```

2. Find the directors who directed the highest number of movies.

```
director_count_df = movies_df.filter(movies_df.directors.isNotNull()) \
                    .groupBy("directors").count() \
                    .orderBy("count", ascending=False)

director_count_df.show(1)
```

```
24/11/14 13:53:01 WARN Native
+--------------+-----+
|     directors|count|
+--------------+-----+
|[Woody Allen]|   39|
+--------------+-----+
only showing top 1 row
```

3.Determine the genres with the highest average IMDb rating (use the imdb.rating field).

```
from pyspark.sql.functions import explode, col, avg

average_rating_df = movies_df \
    .withColumn("genre", explode("genres")) \
    .withColumn("rating", col("imdb.rating.$numberDouble").cast("double")) \
    .groupBy("genre") \
    .agg(avg("rating").alias("avg_rating"))\
    .orderBy('avg_rating', ascending= False)

average_rating_df.show(1)
```

```
24/11/14 14:10:24 WARN NativeCodeLoader: Un
+---------+------------------+
|    genre|        avg_rating|
+---------+------------------+
|Film-Noir|7.5041237113402035|
+---------+------------------+
only showing top 1 row
```

4.Find the month with the most movie releases based on the released date.

```
from pyspark.sql.functions import to_date, month, col

movies_df = movies_df.withColumn("release_date", to_date((col("released.$date.$numberLong").cast("long") / 1000).cast("timestamp")))


movies_df = movies_df.withColumn("release_month", month("release_date"))


month_count_df = movies_df.where(col("release_date").isNotNull()) \
                          .groupBy("release_month") \
                          .count() \
                          .orderBy("count", ascending=False)


month_count_df.show(1)
```

```
+-------------+-----+
|release_month|count|
+-------------+-----+
|           10| 2475|
+-------------+-----+
only showing top 1 row
```

5. Identify the top 5 movies with the longest runtime in each genre.

```
movies_df = movies_df.withColumn("genre", explode("genres"))
movies_df.createOrReplaceTempView("movies")

query = """
SELECT genre, title, runtime
FROM (
    SELECT genre, title, runtime,
           ROW_NUMBER() OVER (PARTITION BY genre ORDER BY runtime DESC) as rank
    FROM movies
) ranked_movies
WHERE rank <= 5
ORDER BY genre, rank
"""

top_5_movies_by_genre = spark.sql(query)
top_5_movies_by_genre.show()
```

```
To adjust logging level use sc.setLogLevel
24/11/14 14:52:42 WARN NativeCodeLoader: Un
+---------+--------------------+-------+
|    genre|               title|runtime|
+---------+--------------------+-------+
|   Action|A Fistful of Dollars|   {99}|
|   Action|              Hooper|   {99}|
|   Action|Chissè perchè... ...|   {99}|
|   Action|Escape from New York|   {99}|
|   Action|           Megaforce|   {99}|
|Adventure|If It's Tuesday, ...|   {99}|
|Adventure|The Island of Dr....|   {99}|
|Adventure|The Sword and the...|   {99}|
|Adventure|          Dreamscape|   {99}|
|Adventure|Big Trouble in Li...|   {99}|
|Animation| The Prince of Egypt|   {99}|
|Animation|         Waking Life|   {99}|
|Animation|Azur & Asmar: The...|   {99}|
|Animation|        King of Jazz|   {98}|
|Animation|The Bugs Bunny/Ro...|   {98}|
|Biography|     Queen Christina|   {99}|
|Biography|Blossoms in the Dust|   {99}|
|Biography|The Positively Tr...|   {99}|
|Biography|  Heavenly Creatures|   {99}|
|Biography|     Dangerous Minds|   {99}|
+---------+--------------------+-------+
only showing top 20 rows
```

6. Find the top 10 actors who appeared in the most movies. For each actor, list the number of movies and their average IMDb rating.

```
from pyspark.sql.functions import explode, avg, count

movies_df = movies_df.withColumn("rating_float", col("imdb.rating.$numberDouble").cast("doubl
movies_df = movies_df.withColumn("actor", explode("cast"))
result_df = movies_df.groupBy("actor")\
    .agg(count("title").alias("movie_count"), avg("rating_float").alias("avg_rating"))\
    .orderBy(col("movie_count").desc())

top_10 = result_df.limit(10)
top_10.show()
```

```
+--------------------+-----------+------------------+
|               actor|movie_count|        avg_rating|
+--------------------+-----------+------------------+
|    Gèrard Depardieu|         68|          6.659375|
|      Robert De Niro|         60| 6.958490566037736|
|       Michael Caine|         53| 6.604347826086958|
|Marcello Mastroianni|         50|7.1688888888888895|
|       Max von Sydow|         49| 6.940000000000001|
|        Bruce Willis|         49|6.4468085106382995|
|      Morgan Freeman|         48| 7.172727272727273|
|   Samuel L. Jackson|         48| 6.390476190476191|
| Christopher Plummer|         47| 7.017948717948718|
|        Gene Hackman|         46| 6.757142857142856|
+--------------------+-----------+------------------+
```

7.Plot a bar chart with the number of movies released each year.

```
from pyspark.sql import functions as F

movies_df = movies_df.withColumn("year", F.regexp_extract("year", r'"(\d+)"', 1).cast("int"))
movies_df = movies_df.filter(F.col("year").isNotNull())
movies_by_year = movies_df.groupBy("year").agg(F.count("title").alias("movie_count"))
movies_by_year_pd = movies_by_year.orderBy("year").toPandas()

import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.bar(movies_by_year_pd["year"], movies_by_year_pd["movie_count"])
plt.xlabel("Year")
plt.ylabel("Number of Movies Released")
plt.title("Number of Movies Released Each Year")
plt.xticks(rotation=45)
plt.show()
```