2024-11-18

DATABASES 2

LAB 8 - MONGODB

Paulina Czarnota C21365726

EXERCISE 1 - SONGS AND SINGERS COLLECTION

INTRODUCTION

This exercise focuses on managing a collection of songs and their associated singers using MongoDB. Tasks include inserting data, performing aggregations, and querying the database using MongoDB's aggregation framework operators such as $project, $unwind, and $group.

Step 1: Setup and Data Insertion

1. Start MongoDB Shell:

mongosh

2. Create the Database:

use lab8;

3. Insert Data Using the Script (singer.js):

```
switched to db lab8
lab8> db.songs.insertMany([
...   {
...     songId: 1,
...     title: "Shape of You",
...     singer: { name: "Ed Sheeran", age: 32 },
...     album: "Divide",
...     releaseYear: 2017,
...     genres: ["Pop", "Dance"],
...     duration: 233, // in seconds
...     ratings: [5, 4, 5, 5, 4]
...   },
...   {
...     songId: 2,
...     title: "Blinding Lights",
...     singer: { name: "The Weeknd", age: 33 },
...     album: "After Hours",
...     releaseYear: 2020,
...     genres: ["Synthwave", "Pop"],
...     duration: 200,
...     ratings: [5, 5, 4, 4, 5]
...   },
...   {
...     songId: 3,
...     title: "Someone Like You",
...     singer: { name: "Adele", age: 35 },
...     album: "21",
...     releaseYear: 2011,
...     genres: ["Soul", "Pop"],
...     duration: 285,
...     ratings: [5, 5, 5, 4, 5]
...   },
...   {
...     songId: 4,
...     title: "Levitating",
...     singer: { name: "Dua Lipa", age: 28 },
...     album: "Future Nostalgia",
...     releaseYear: 2020,
...     genres: ["Pop", "Dance"],
...     duration: 203,
...     ratings: [4, 5, 4, 4, 4]
...   }
... ]);
```

```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6755131323f01a26e6893bf8'),
    '1': ObjectId('6755131323f01a26e6893bf9'),
    '2': ObjectId('6755131323f01a26e6893bfa'),
    '3': ObjectId('6755131323f01a26e6893bfb')
  }
}
```

4. Verify Data Insertion:

```
lab8> db.songs.find().pretty();
[
  {
    _id: ObjectId('67550fae8734830eaf893bf8'),
    songId: 1,
    title: 'Shape of You',
    singer: { name: 'Ed Sheeran', age: 32 },
    album: 'Divide',
    releaseYear: 2017,
    genres: [ 'Pop', 'Dance' ],
    duration: 233,
    ratings: [ 5, 4, 5, 5, 4 ]
  },
  {
    _id: ObjectId('67550fae8734830eaf893bf9'),
    songId: 2,
    title: 'Blinding Lights',
    singer: { name: 'The Weeknd', age: 33 },
    album: 'After Hours',
    releaseYear: 2020,
    genres: [ 'Synthwave', 'Pop' ],
    duration: 200,
    ratings: [ 5, 5, 4, 4, 5 ]
  },
  {
    _id: ObjectId('67550fae8734830eaf893bfa'),
    songId: 3,
    title: 'Someone Like You',
    singer: { name: 'Adele', age: 35 },
    album: '21',
    releaseYear: 2011,
    genres: [ 'Soul', 'Pop' ],
    duration: 285,
    ratings: [ 5, 5, 5, 4, 5 ]
  },
  {
    _id: ObjectId('67550fae8734830eaf893bfb'),
    songId: 4,
    title: 'Levitating',
    singer: { name: 'Dua Lipa', age: 28 },
```

```
lab8>
    releaseYear: 2020,
    genres: [ 'Pop', 'Dance' ],
    duration: 203,
    ratings: [ 4, 5, 4, 4, 4 ]
  },
  {
    _id: ObjectId('6755131323f01a26e6893bf8'),
    songId: 1,
    title: 'Shape of You',
    singer: { name: 'Ed Sheeran', age: 32 },
    album: 'Divide',
    releaseYear: 2017,
    genres: [ 'Pop', 'Dance' ],
    duration: 233,
    ratings: [ 5, 4, 5, 5, 4 ]
  },
  {
    _id: ObjectId('6755131323f01a26e6893bf9'),
    songId: 2,
    title: 'Blinding Lights',
    singer: { name: 'The Weeknd', age: 33 },
    album: 'After Hours',
    releaseYear: 2020,
    genres: [ 'Synthwave', 'Pop' ],
    duration: 200,
    ratings: [ 5, 5, 4, 4, 5 ]
  },
  {
    _id: ObjectId('6755131323f01a26e6893bfa'),
    songId: 3,
    title: 'Someone Like You',
    singer: { name: 'Adele', age: 35 },
    album: '21',
    releaseYear: 2011,
    genres: [ 'Soul', 'Pop' ],
    duration: 285,
    ratings: [ 5, 5, 5, 4, 5 ]
  },
  {
    _id: ObjectId('6755131323f01a26e6893bfb'),
    songId: 4,
    title: 'Levitating',
    singer: { name: 'Dua Lipa', age: 28 },
    album: 'Future Nostalgia',
    releaseYear: 2020,
    genres: [ 'Pop', 'Dance' ],
    duration: 203,
    ratings: [ 4, 5, 4, 4, 4 ]
  }
]
```

Step 2: Queries and Outputs

Query 1: Calculate Average Rating for Each Song

```
lab8> db.songs.aggregate([
...   {
...     $project: {
...       title: 1,
...       averageRating: { $avg: "$ratings" }
...     }
...   }
... ]);
```

Explanation:

$project selects only the title and calculates the average rating using $avg on the ratings array.

Output:

```
[
  {
    _id: ObjectId('67550fae8734830eaf893bf8'),
    title: 'Shape of You',
    averageRating: 4.6
  },
  {
    _id: ObjectId('67550fae8734830eaf893bf9'),
    title: 'Blinding Lights',
    averageRating: 4.6
  },
  {
    _id: ObjectId('67550fae8734830eaf893bfa'),
    title: 'Someone Like You',
    averageRating: 4.8
  },
  {
    _id: ObjectId('67550fae8734830eaf893bfb'),
    title: 'Levitating',
    averageRating: 4.2
  },
  {
    _id: ObjectId('6755131323f01a26e6893bf8'),
    title: 'Shape of You',
    averageRating: 4.6
  },
  {
    _id: ObjectId('6755131323f01a26e6893bf9'),
    title: 'Blinding Lights',
    averageRating: 4.6
  },
  {
    _id: ObjectId('6755131323f01a26e6893bfa'),
    title: 'Someone Like You',
    averageRating: 4.8
  },
  {
    _id: ObjectId('6755131323f01a26e6893bfb'),
    title: 'Levitating',
    averageRating: 4.2
  }
]
```

Query 2: Find Songs Longer Than 3 Minutes (180 Seconds)

```
db.songs.find(
  { duration: { $gt: 180 } },
  { title: 1, duration: 1, _id: 0 }
);
```

```
...   { duration: { $gt: 180 } },
...   { title: 1, duration: 1, _id: 0 }
... );
```

Explanation:

- $gt checks for songs where duration is greater than 180 seconds.
- The query only returns the title and duration fields.

Output:

```
[
  { title: 'Shape of You', duration: 233 },
  { title: 'Blinding Lights', duration: 200 },
  { title: 'Someone Like You', duration: 285 },
  { title: 'Levitating', duration: 203 },
  { title: 'Shape of You', duration: 233 },
  { title: 'Blinding Lights', duration: 200 },
  { title: 'Someone Like You', duration: 285 },
  { title: 'Levitating', duration: 203 }
]
```

Query 3: Count Songs Per Genre

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

```
...   { $unwind: "$genres" },
...   { $group: { _id: "$genres", count: { $sum: 1 } } },
...   { $sort: { count: -1 } }
... ]);
```

Explanation:

- $unwind expands the genres array into separate records.

- $group counts each occurrence of genres.
- $sort orders the genres by count in descending order.

Output:

```
[
  { _id: 'Pop', count: 8 },
  { _id: 'Dance', count: 4 },
  { _id: 'Soul', count: 2 },
  { _id: 'Synthwave', count: 2 }
]
```

Query 4: Find the Oldest Singer

```
db.songs.aggregate([
  { $sort: { "singer.age": -1 } },
  { $limit: 1 },
  { $project: { _id: 0, "singer.name": 1, "singer.age": 1 } }
]);
```

```
...   { $sort: { "singer.age": -1 } },
...   { $limit: 1 },
...   { $project: { _id: 0, "singer.name": 1, "singer.age": 1 } }
... ]);
```

Explanation:

- $sort orders singers by age in descending order.
- $limit returns only the first result.
- $project specifies only the singer's name and age.

Output:

```
[ { singer: { name: 'Adele', age: 35 } } ]
```

Query 5: List Songs Released After 2015

```
db.songs.find(
  { releaseYear: { $gt: 2015 } },
  { title: 1, "singer.name": 1, releaseYear: 1, _id: 0 }
);
```

```
lab8> db.songs.find(
...   { releaseYear: { $gt: 2015 } },
...   { title: 1, "singer.name": 1, releaseYear: 1, _id: 0 }
... );
```

Explanation:

- $gt filters songs where releaseYear is after 2015.
- The query only includes title, singer.name, and releaseYear fields.

Output:

```
[
  {
    title: 'Shape of You',
    singer: { name: 'Ed Sheeran' },
    releaseYear: 2017
  },
  {
    title: 'Blinding Lights',
    singer: { name: 'The Weeknd' },
    releaseYear: 2020
  },
  {
    title: 'Levitating',
    singer: { name: 'Dua Lipa' },
    releaseYear: 2020
  },
  {
    title: 'Shape of You',
    singer: { name: 'Ed Sheeran' },
    releaseYear: 2017
  },
  {
    title: 'Blinding Lights',
    singer: { name: 'The Weeknd' },
    releaseYear: 2020
  },
  {
    title: 'Levitating',
    singer: { name: 'Dua Lipa' },
    releaseYear: 2020
  }
]
```

Query 6: Calculate Total Duration of Songs Per Singer

```
db.songs.aggregate([
  {
    $group: {
      _id: "$singer.name",
      totalDuration: { $sum: "$duration" }
    }
  },
  { $sort: { totalDuration: -1 } }
```

]);

```
...   {
...     $group: {
...       _id: "$singer.name",
...       totalDuration: { $sum: "$duration" }
...     }
...   },
...   { $sort: { totalDuration: -1 } }
... ]);
```

Explanation:

- $group groups records by singer.name and calculates the total duration using $sum.
- $sort orders by totalDuration in descending order.

Output:

```
[
  { _id: 'Adele', totalDuration: 570 },
  { _id: 'Ed Sheeran', totalDuration: 466 },
  { _id: 'Dua Lipa', totalDuration: 406 },
  { _id: 'The Weeknd', totalDuration: 400 }
]
```

Query 7: Find Songs with Multiple Genres

db.songs.aggregate([

{ $project: { title: 1, genreCount: { $size: "$genres" } } },

{ $match: { genreCount: { $gt: 1 } } }

]);

```
...   { $project: { title: 1, genreCount: { $size: "$genres" } } },
...   { $match: { genreCount: { $gt: 1 } } }
... ]);
```

Explanation:

- $project creates a new field genreCount as the size of the genres array.
- $match filters songs with more than one genre.

Output:

Query 8: Group Songs by Release Year

```
db.songs.aggregate([
  {
    $group: {
      _id: "$releaseYear",
      songs: { $push: "$title" }
    }
  },
  { $sort: { _id: 1 } }
]);
```

```
...   {
...     $group: {
...       _id: "$releaseYear",
...       songs: { $push: "$title" }
...     }
...   },
...   { $sort: { _id: 1 } }
... ]);
```

Explanation:

- $group organizes songs by releaseYear and pushes song titles into an array.
- $sort orders results by releaseYear in ascending order.

Output:

```
[
  { _id: 2011, songs: [ 'Someone Like You', 'Someone Like You' ] },
  { _id: 2017, songs: [ 'Shape of You', 'Shape of You' ] },
  {
    _id: 2020,
    songs: [
      'Blinding Lights',
      'Levitating',
      'Blinding Lights',
      'Levitating'
    ]
  }
]
```

CONCLUSION

These queries demonstrate the use of MongoDB's aggregation pipeline and query operators for managing songs and singers effectively

EXERCISE 2 - STUDENTS, COURSES, AND MARKS COLLECTION

INTRODUCTION

The objective of this exercise is to create a MongoDB collection by embedding JSON objects representing relational tables: Students, Courses, and Marks. Queries will extract meaningful insights such as finding students who failed, counting exam passes, and determining the highest average mark.

Step 1: Data Insertion

1. Open MongoDB Shell:

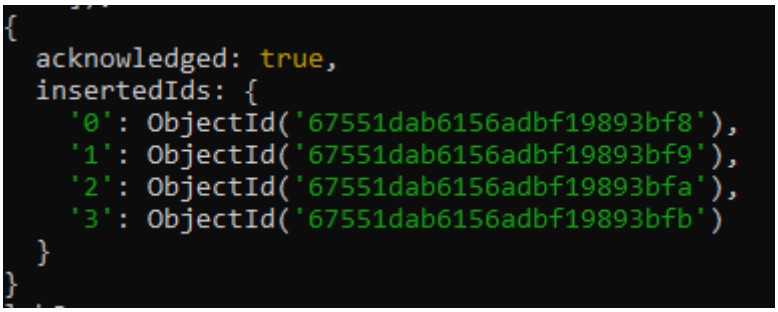
mongosh

2. Create the Database:

use lab8;

3. Insert the Data into MongoDB:

```
switched to db lab8
lab8> db.students_courses_marks.insertMany([
...   {
...     studentId: 1,
...     name: "Mary",
...     surname: "Murray",
...     nationality: "Irish",
...     age: 45,
...     marks: [
...       { courseId: "DB", courseName: "Databases", credits: 10, mark: 56, examDate: "2011-10-10" },
...       { courseId: "MA", courseName: "Maths", credits: 5, mark: 76, examDate: "2012-11-09" },
...       { courseId: "PR", courseName: "Programming", credits: 15, mark: 45, examDate: "2014-07-02" }
...     ]
...   },
...   {
...     studentId: 2,
...     name: "Bill",
...     surname: "Bellichick",
...     nationality: "American",
...     age: 32,
...     marks: [
...       { courseId: "DB", courseName: "Databases", credits: 10, mark: 55, examDate: "2011-10-10" },
...       { courseId: "MA", courseName: "Maths", credits: 5, mark: 87, examDate: "2012-11-09" },
...       { courseId: "PR", courseName: "Programming", credits: 15, mark: 45, examDate: "2011-10-10" }
...     ]
...   },
...   {
...     studentId: 3,
...     name: "Tom",
...     surname: "Brady",
...     nationality: "Canadian",
...     age: 22,
...     marks: [
...       { courseId: "DB", courseName: "Databases", credits: 10, mark: 34, examDate: "2012-11-09" },
...       { courseId: "MA", courseName: "Maths", credits: 5, mark: 56, examDate: "2014-07-02" }
...     ]
...   },
...   {
...     studentId: 4,
...     name: "John",
...     surname: "Bale",
...     nationality: "English",
...     age: 24,
...     marks: [
...       { courseId: "DB", courseName: "Databases", credits: 10, mark: 71, examDate: "2011-10-10" },
...       { courseId: "MA", courseName: "Maths", credits: 5, mark: 88, examDate: "2011-10-10" },
...       { courseId: "PR", courseName: "Programming", credits: 15, mark: 95, examDate: "2012-11-09" }
...     ]
...   }
... ]);
```



4. Verify Data Insertion:

db.students_courses_marks.find().pretty();

Output:

Full records for all inserted students, matching the insertion script.

Step 2: Queries and Outputs

Query 1: Find Students Who Failed (Mark < 50)

```
db.students_courses_marks.aggregate([
  { $unwind: "$marks" },
  { $match: { "marks.mark": { $lt: 50 } } },
  {
    $project: {
      _id: 0,
      name: 1,
      surname: 1,
      failedCourse: "$marks.courseName",
      mark: "$marks.mark"
    }
  }
]);
```

```
...     { $unwind: "$marks" },
...     { $match: { "marks.mark": { $lt: 50 } } },
...     {
...       $project: {
...         _id: 0,
...         name: 1,
...         surname: 1,
...         failedCourse: "$marks.courseName",
...         mark: "$marks.mark"
...       }
...     }
... ]);
```

Explanation:

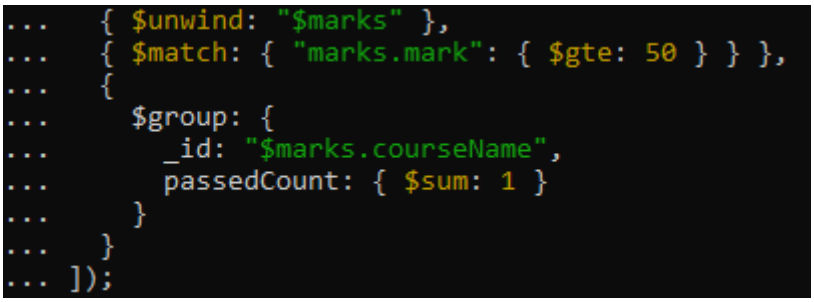
- $unwind: Expands the marks array into individual records.
- $match: Filters only records where mark < 50.
- $project: Displays only relevant fields (name, surname, failedCourse, and mark).

Output:

```
[
  {
    name: 'Mary',
    surname: 'Murray',
    failedCourse: 'Programming',
    mark: 45
  },
  {
    name: 'Bill',
    surname: 'Bellichick',
    failedCourse: 'Programming',
    mark: 45
  },
  {
    name: 'Tom',
    surname: 'Brady',
    failedCourse: 'Databases',
    mark: 34
  },
  {
    name: 'Mary',
    surname: 'Murray',
    failedCourse: 'Programming',
    mark: 45
  },
  {
    name: 'Bill',
    surname: 'Bellichick',
    failedCourse: 'Programming',
    mark: 45
  },
  {
    name: 'Tom',
    surname: 'Brady',
    failedCourse: 'Databases',
    mark: 34
  },
  {
    name: 'Mary',
    surname: 'Murray',
    failedCourse: 'Programming',
    mark: 45
  },
  {
    name: 'Bill',
    surname: 'Bellichick',
    failedCourse: 'Programming',
    mark: 45
  },
  {
    name: 'Tom',
    surname: 'Brady',
    failedCourse: 'Databases',
    mark: 34
  }
]
```

Query 2: Count Students Who Passed Each Exam (Mark >= 50)

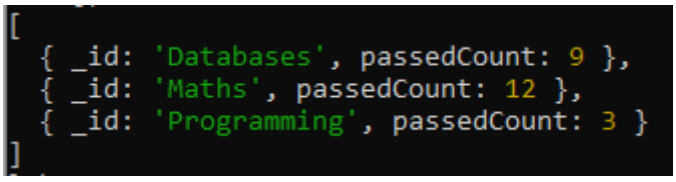
```
db.students_courses_marks.aggregate([
  { $unwind: "$marks" },
  { $match: { "marks.mark": { $gte: 50 } } },
  {
   $group: {
    _id: "$marks.courseName",
    passedCount: { $sum: 1 }
   }
  }
]);
```



Explanation:

- $unwind: Expands the marks array into individual documents.
- $match: Filters only marks greater than or equal to 50.
- $group: Groups records by courseName and counts the number of passing students.

Output:



Query 3: Find the Student with the Highest Average Mark

```
db.students_courses_marks.aggregate([
  { $unwind: "$marks" },
  {
```

```
  $group: {
    _id: { name: "$name", surname: "$surname" },
    averageMark: { $avg: "$marks.mark" }
  }
 },
 { $sort: { averageMark: -1 } },
 { $limit: 1 }
])
```

```
...   { $unwind: "$marks" },
...   {
...     $group: {
...       _id: { name: "$name", surname: "$surname" },
...       averageMark: { $avg: "$marks.mark" }
...     }
...   },
...   { $sort: { averageMark: -1 } },
...   { $limit: 1 }
... ])
```

Explanation:

- $unwind: Expands the marks array into individual records.
- $group: Groups records by student name and surname, calculating the average mark.
- $sort: Orders results by averageMark in descending order.
- $limit: Limits the result to only one student.

Output:

```
[
  {
    _id: { name: 'John', surname: 'Bale' },
    averageMark: 84.66666666666667
  }
]
```

CONCLUSION

This exercise demonstrated MongoDB schema design, data insertion, and querying using aggregation pipelines. Key tasks included identifying failing students, counting passed exams, and determining the student with the highest average mark. These tasks reinforce NoSQL database management skills using MongoDB's aggregation framework.