

## Task 7: Advanced Aggregation with MongoDB

### Objective:

Understand and practice advanced aggregation techniques in MongoDB, including using advanced aggregation operators (lookup, unwind, group) and pipeline optimization.

### Prerequisites:

- Basic understanding of JavaScript and MongoDB.
- Node.js installation.
- MongoDB installed and running.
- A MongoDB collection with sample movie data and a collection with production company data.

### Concepts:

#### 1. Advanced Aggregation Operators:

##### \$lookup Operator:

- The \$lookup operator performs a left outer join to a collection in the same database to filter in documents from the "joined" collection for processing.

##### Example:

##### JavaScript:

```
const lookup = await collection.aggregate([
  {
    $lookup: {
      from: 'production_companies',
      localField: 'title',
      foreignField: 'movies',
      as: 'production_companies'
    }
  }
]).toArray();
console.log('Movies with Production Companies:', lookup);
```

##### MongoDB Compass:

- Open MongoDB Compass.
- Connect to your MongoDB instance.
- Select your database and collection.
- Click on the Aggregation tab.
- Add a new stage and select \$lookup .
- Enter the following condition:

```
{
  "from": "production_companies",
  "localField": "title",
  "foreignField": "movies",
  "as": "production_companies"
}
```

- Click on the Add Stage button.

### **\$unwind Operator:**

- The \$unwind operator deconstructs an array field from the input documents to output a document for each element.

#### **Example:**

#### **JavaScript:**

```
const unwind = await collection.aggregate([
  { $unwind: "$production_companies" }
]).toArray();
console.log('Unwind Production Companies:', unwind);
```

### **MongoDB Compass:**

- Add a new stage and select \$unwind .
- Enter the following condition:

```
{ "path": "$production_companies" }
```

- Click on the Add Stage button.

### **\$group Operator:**

- The \$group operator groups input documents by the specified \_id expression and for each distinct grouping, outputs a document.
- The output documents can also contain computed fields that hold the values of some accumulator expressions.

#### **Example:**

#### **JavaScript:**

```
const group = await collection.aggregate([
  { $group: { _id: "$genre", averagePopularity: { $avg: "$popularity" } } }
]).toArray();
console.log('Grouped Movies:', group);
```

### **MongoDB Compass:**

- Add a new stage and select \$group .
- Enter the following condition:

```
{ "_id": "$genre", "averagePopularity": { "$avg": "$popularity" } } }
```

- Click on the Add Stage button.

## 2. Pipeline Optimization:

- Pipeline optimization involves arranging stages in an efficient manner to reduce the amount of data processed in each stage.
- Use \$match and \$project stages early in the pipeline to filter out unnecessary data.

### Example:

#### JavaScript:

```
const optimizedPipeline = await collection.aggregate([
  { $match: { popularity: { $gt: 50 } } },
  { $project: { title: 1, genre: 1, popularity: 1 } },
  {
    $lookup: {
      from: 'production_companies',
      localField: 'title',
      foreignField: 'movies',
      as: 'production_companies'
    }
  },
  { $unwind: "$production_companies" },
  { $group: { _id: "$genre", averagePopularity: { $avg: "$popularity" } } },
  { $sort: { averagePopularity: -1 } }
]).toArray();
console.log('Optimized Pipeline Results:', optimizedPipeline);
```

#### MongoDB Compass:

- Add a new stage and select \$match .
- Enter the following condition:

```
{ "popularity": { "$gt": 50 } }
```

- Click on the Add Stage button.
- Add a new stage and select \$project .
- Enter the following condition:

```
{ "title": 1, "genre": 1, "popularity": 1 }
```

- Click on the Add Stage button.


### Instructions:

Perform the following queries:

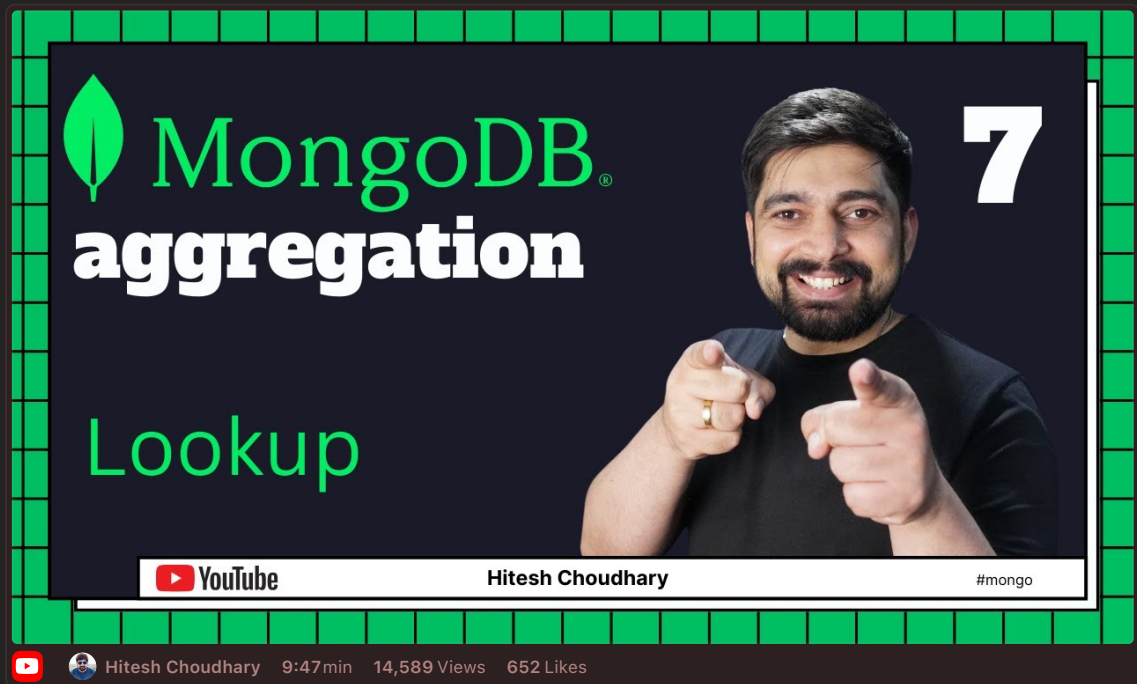
1. Use the `$lookup` operator to join the `movies` collection with the `production companies` collection based on `title`.
2. Use the `$unwind` operator to deconstruct the `production_companies` array field.
3. Use the `$group` operator to group movies by genre and calculate the average popularity.
4. Optimize the pipeline by using `$match` and `$project` stages early to filter and limit data processing.

## Resources:

### Documentation:

-  **Aggregation Operations**  
Aggregation operations process multiple documents and return computed — results...  
 <https://docs.mongodb.com/manual/aggregation/>
-  **\$lookup (aggregation)**  
Performs a left outer join to a collection in the same database to — filter in documen...  
 <https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/>
-  **\$unwind (aggregation)**  
Deconstructs an array field from the input documents to output a — document for e...  
 <https://docs.mongodb.com/manual/reference/operator/aggregation/unwind/>
-  **\$group (aggregation)**  
The `$group` stage separates documents into groups according to a — "group key". T...  
 <https://docs.mongodb.com/manual/reference/operator/aggregation/group/>
-  **Aggregation Pipeline Optimization**  
Aggregation pipeline operations have an optimization phase which — attempts to re...  
 <https://docs.mongodb.com/manual/core/aggregation-pipeline-optimization/>

## Videos:



## GitHub Instructions

### Open in Visual Studio Code:

After clicking on the "Open in Visual Studio Code" button from the GitHub Classroom confirmation page, Visual Studio Code (VSCode) will open the repository directly.

If prompted, select "Open" or "Allow" to open the repository in VSCode.

### Complete the Task:

In VSCode, open the `index.js` file in the root directory of your repository and write your solution.

Ensure the `package.json` file is present and contains all necessary dependencies. If you need to install additional packages, use:

```
npm i
```

### Run and Test Your Code:

Run your code to ensure it works correctly. Use the following command:

```
node index.js
```

### Commit Your Changes:

Commit your changes with a meaningful message:

```
git commit -m "Completed task 7"
```

### Push Your Changes to Your Forked Repository:

Push your changes to your forked repository:

```
git push origin main
```

### Create a Pull Request:

Go to your forked repository on GitHub.

Click on the "Pull Requests" tab.

Click the "New Pull Request" button.

Ensure the base repository is the original template repository and the base branch is `main`.

Ensure the head repository is your forked repository and the compare branch is `main`.

Click "Create Pull Request".

Add a title and description for your pull request and submit it.

### Summary of Commands

```
# Fork the repository on GitHub

# Clone the forked repository
git clone https://github.com/your-github-username/repository-name.git
cd repository-name

# Complete the task by editing index.js

# Run your code
node index.js

# Add, commit, and push your changes
git add index.js package.json
git commit -m "Completed task 7"
git push origin main

# Create a pull request on GitHub
```