### Problem Statement: Create Collection `Movies\_Data` and Perform Queries

We will create a collection `Movies\_Data` in MongoDB with the fields `Movie\_ID`, `Movie\_Name`, `Director`, `Genre`, and `BoxOfficeCollection`. After creating the collection, we will execute various queries and indexing tasks as described.

---

### Step 1: Create the Collection `Movies\_Data` and Insert Documents

First, create the collection and insert some sample documents. Here's how you can do it:

```javascript

use movie\_db // Switch to the desired database

db.Movies\_Data.insertMany([

{ Movie\_ID: 1, Movie\_Name: "The Dark Knight", Director: "Christopher Nolan", Genre: "Action", BoxOfficeCollection: 1000000000 },

{ Movie\_ID: 2, Movie\_Name: "Inception", Director: "Christopher Nolan", Genre: "Sci-Fi", BoxOfficeCollection: 800000000 },

{ Movie\_ID: 3, Movie\_Name: "Interstellar", Director: "Christopher Nolan", Genre: "Sci-Fi", BoxOfficeCollection: 700000000 },

{ Movie\_ID: 4, Movie\_Name: "The Prestige", Director: "Christopher Nolan", Genre: "Thriller", BoxOfficeCollection: 500000000 },

{ Movie\_ID: 5, Movie\_Name: "The Lion King", Director: "Jon Favreau", Genre: "Animation", BoxOfficeCollection: 1600000000 },

{ Movie\_ID: 6, Movie\_Name: "Avengers: Endgame", Director: "Anthony Russo", Genre: "Action", BoxOfficeCollection: 2500000000 },

{ Movie\_ID: 7, Movie\_Name: "Guardians of the Galaxy", Director: "James Gunn", Genre: "Action", BoxOfficeCollection: 800000000 },

{ Movie\_ID: 8, Movie\_Name: "Frozen II", Director: "Chris Buck", Genre: "Animation", BoxOfficeCollection: 1300000000 },

{ Movie\_ID: 9, Movie\_Name: "Star Wars: The Force Awakens", Director: "J.J. Abrams", Genre: "Sci-Fi", BoxOfficeCollection: 2000000000 },

{ Movie\_ID: 10, Movie\_Name: "The Matrix", Director: "Lana Wachowski", Genre: "Sci-Fi", BoxOfficeCollection: 460000000 }

]);

```

---

### Step 2: Queries for Aggregation Operations

#### 1. Display a List Stating How Many Movies Are Directed by Each "Director"

We can use the `$group` stage to group movies by `Director` and count the number of movies directed by each one.

```javascript

db.Movies\_Data.aggregate([

{

$group: {

\_id: "$Director", // Group by Director

movieCount: { $sum: 1 } // Count number of movies per director

}

}

]).pretty();

```

- \*\*Explanation\*\*: This query groups the movies by `Director` and uses `$sum: 1` to count the number of movies each director has made.

#### 2. Display List of Movies with the Highest BoxOfficeCollection in Each Genre

We can use the `$group` stage to find the highest `BoxOfficeCollection` for each genre and return the corresponding movie name.

```javascript

db.Movies\_Data.aggregate([

{

$group: {

\_id: "$Genre", // Group by Genre

highestBoxOfficeMovie: { $first: "$Movie\_Name" }, // Get the movie with the highest BoxOfficeCollection

highestBoxOfficeCollection: { $max: "$BoxOfficeCollection" } // Get the highest BoxOfficeCollection in each genre

}

}

]).pretty();

```

- \*\*Explanation\*\*: This query groups the movies by `Genre`, and then uses `$first` to retrieve the movie name with the highest box office collection for each genre, and `$max` to get the highest collection value.

#### 3. Display List of Movies with the Highest BoxOfficeCollection in Each Genre in Ascending Order of BoxOfficeCollection

We will first sort the movies by `BoxOfficeCollection` in descending order, then group them by genre to find the highest-grossing movie in each genre.

```javascript

db.Movies\_Data.aggregate([

{

$sort: { BoxOfficeCollection: -1 } // Sort movies by BoxOfficeCollection in descending order

},

{

$group: {

\_id: "$Genre", // Group by Genre

highestBoxOfficeMovie: { $first: "$Movie\_Name" }, // Get the highest-grossing movie in each genre

highestBoxOfficeCollection: { $first: "$BoxOfficeCollection" } // Get the highest BoxOfficeCollection

}

},

{

$sort: { highestBoxOfficeCollection: 1 } // Sort by BoxOfficeCollection in ascending order

}

]).pretty();

```

- \*\*Explanation\*\*: The `$sort` stage first orders the movies by `BoxOfficeCollection` in descending order. Then, `$group` is used to find the highest-grossing movie per genre. Finally, the last `$sort` ensures that the movies are displayed in ascending order of `BoxOfficeCollection`.

---

### Step 3: Create Indexes

#### 4. Create an Index on the Field `Movie\_ID`

We can create a single field index on `Movie\_ID` to optimize queries that search by this field.

```javascript

db.Movies\_Data.createIndex({ Movie\_ID: 1 });

```

- \*\*Explanation\*\*: This creates an ascending index on the `Movie\_ID` field.

#### 5. Create an Index on Fields `Movie\_Name` and `Director`

For faster queries that search by both `Movie\_Name` and `Director`, we can create a composite index.

```javascript

db.Movies\_Data.createIndex({ Movie\_Name: 1, Director: 1 });

```

- \*\*Explanation\*\*: This creates a compound index on the fields `Movie\_Name` and `Director`, both in ascending order.

---

### Step 4: Drop Indexes

#### 6. Drop the Index on the `Movie\_ID` Field

If the index on `Movie\_ID` is no longer needed, we can drop it using the following command:

```javascript

db.Movies\_Data.dropIndex({ Movie\_ID: 1 });

```

- \*\*Explanation\*\*: This drops the index on the `Movie\_ID` field.

#### 7. Drop the Index on Fields `Movie\_Name` and `Director`

To drop the composite index on `Movie\_Name` and `Director`, use the following command:

```javascript

db.Movies\_Data.dropIndex({ Movie\_Name: 1, Director: 1 });

```

- \*\*Explanation\*\*: This removes the composite index on the `Movie\_Name` and `Director` fields.

---

### Step 5: Additional Index Management

#### Show All Indexes

To view all the indexes created on the `Movies\_Data` collection, run:

```javascript

db.Movies\_Data.getIndexes();

```

- \*\*Explanation\*\*: This command lists all indexes on the `Movies\_Data` collection, including the default `\_id` index and any custom indexes you have created.

---

### Summary of Queries and Operations

1. \*\*Display the count of movies directed by each director\*\*: Uses `$group` with `$sum`.

2. \*\*List the highest box office collection movie in each genre\*\*: Uses `$group` with `$max` and `$first`.

3. \*\*List the highest box office collection movie in each genre in ascending order of box office collection\*\*: Uses `$sort`, `$group`, and `$sort` again.

4. \*\*Create an index on `Movie\_ID`\*\*: Uses `createIndex()` on `Movie\_ID`.

5. \*\*Create a composite index on `Movie\_Name` and `Director`\*\*: Uses `createIndex()` on both fields.

6. \*\*Drop the index on `Movie\_ID`\*\*: Uses `dropIndex()` on `Movie\_ID`.

7. \*\*Drop the composite index on `Movie\_Name` and `Director`\*\*: Uses `dropIndex()` on both fields.

8. \*\*View all indexes on the collection\*\*: Uses `getIndexes()`.

---

### Running the Queries

1. \*\*Open MongoDB Shell\*\* (`mongosh`).

2. \*\*Switch to the desired database\*\*:

```javascript

use movie\_db

```

3. \*\*Execute the queries\*\*: Run each of the queries listed above.

By following these steps, you can efficiently manage the `Movies\_Data` collection and perform aggregation and indexing operations to retrieve important insights from the data.