"For only movies older than 2001, find the average and maximum popularity for each genre, sort the genres by popularity, and find the adjusted (with trailers) runtime of the longest movie in each genre."

Translate the query into sequential stages so that you can map to your aggregation stages:

- Match movies that were released before 2001.
- Find the average popularity of each genre.
- Sort the genres by popularity.
- Output the adjusted runtime of each movie.

```
var findGenrePopularity = function() {
   print("Finding popularity of each genre");
   var pipeline = [
      { $match: {
      released: {$lte: new ISODate("2001-01-01T00:00:00Z") }}},
      { $group: {
         _id: {"$arrayElemAt": ["$genres", 0]},
         "popularity": { $avg: "$imdb.rating"},
         "top_movie": { $max: "$imdb.rating"},
         "longest_runtime": { $max: "$runtime"}
      }},
         { $sort: { popularity: -1}},
         { $project: {
            _id: 1,
            popularity: 1,
            top_movie: 1,
            adjusted_runtime: { $add: [ "$longest_runtime", 12 ] } } }
      ];
      db.movies.aggregate(pipeline).forEach(printjson);
   }
   findGenrePopularity();
```

The $arrayElemAt takes an element from an array at the specified index (in this case, 0), i.e., primary genre.

You add 12 minutes because the client (the cinema company) has informed you that this is the length of the trailers running before each movie. Once you have the adjusted runtime, you will no longer need longest_runtime

Working with Large Datasets

discuss a few of the aggregation stages that you need to master when working with large, multi-collection datasets.

Sampling with $sample (random sampling)

```
var findWithSample = function() {
   print("Finding all documents WITH sampling")
   var now = Date.now();
   var pipeline = [
      { $sample: {size: 100}},
      { $match: {
         "plot": { $regex: /around/}
      }}
   ];
   db.movies.aggregate(pipeline)
   var duration = Date.now() - now;
   print("Finished WITH sampling in " + duration+"ms");
}
findWithSample();

var findWithoutSample = function() {
   print("Finding all documents WITHOUT sampling")
   var now = Date.now();
   var pipeline =[
      { $match: {
         "plot": { $regex: /around/}
      }},
   ]
   db.movies.aggregate(pipeline)
   var duration = Date.now() - now;
   print("Finished WITHOUT sampling in " + duration+ "ms");
}
findWithoutSample();
```

Joining Collections with $lookup

```
var lookupExample = function() {
   var pipeline = [
      { $match: { $or: [{"name": "Catelyn Stark"}, {"name": "Ned Stark"}]}},
      { $lookup: {
         from: "comments",
         localField: "name",
         foreignField: "name",
         as: "comments"
      }},
  { $limit: 2},
   ];
   db.users.aggregate(pipeline).forEach(printjson);
}
lookupExample();
```

the lookup takes the name of our user, searches the **`comments`** collection, and adds any comments with the same name into a new array field for the original user document. This new array is called **comments**.

$unwind is a relatively simple stage. It deconstructs an array field from an input document to output a new document for each element in the array.

```
var lookupExample = function() {
   var pipeline = [
      { $match: { $or: [{"name": "Catelyn Stark"}, {"name": "Ned Stark"}]}},
      { $lookup: {
         from: "comments",
         localField: "name",
         foreignField: "name",
         as: "comments"
      }},
      { $unwind: "$comments"},
      { $limit: 3},
   ];
   db.users.aggregate(pipeline).forEach(printjson);
}
lookupExample();
```

Outputting Your Results with $out and $merge

```
var findTopRomanceMovies = function() {
   var pipeline = [
      { $sort: {"imdb.rating": -1}}, // Sort by IMDB rating.
      { $match: {
         genres: {$in: ["Romance"]}, // Romance movies only.
         released: {$lte: new ISODate("2001-01-01T00:00: 00Z") }}},
      { $limit: 5 },                // Limit to 5 results.
      { $project: { title: 1, genres: 1, released: 1, "imdb.rating": 1}},
      { $out: "movies_top_romance"}
   ];
   db.movies.aggregate(pipeline).forEach(printjson);
}
findTopRomanceMovies();
```

obtain a list of movies that generate the most comments from users

```
// Ch7_Exercise5.js
var findMostCommentedMovies = function() {
   print("Finding the most commented on movies.");
   var pipeline = [
         { $sample: {}},
         { $group: {}},
         { $sort: {}},
         { $limit: 5},
```

```
        { $lookup: {}},
        { $unwind: },
        { $project: {}},
        { $out: {}}
    ];
    db.comments.aggregate(pipeline).forEach(printjson);
}
findMostCommentedMovies();


// Ch7_Exercise5.js
var findMostCommentedMovies = function() {
    print("Finding the most commented on movies.");
    var pipeline = [
        { $sample: {size: 5000}},
        { $group: {
            _id: "$movie_id",
            "sumComments": { $sum: 1}
        }},
        { $sort: { "sumComments": -1}},
        { $limit: 5},
        { $lookup: {
            from: "movies",
            localField: "_id",
            foreignField: "_id",
            as: "movie"
        }},
        { $unwind: "$movie" },
        { $project: {
            "movie.title": 1,
            "movie.imdb.rating": 1,
            "sumComments": 1,
        }},
        { $out: "most_commented_movies" }
    ];
    db.comments.aggregate(pipeline).forEach(printjson);
}
findMostCommentedMovies();
```