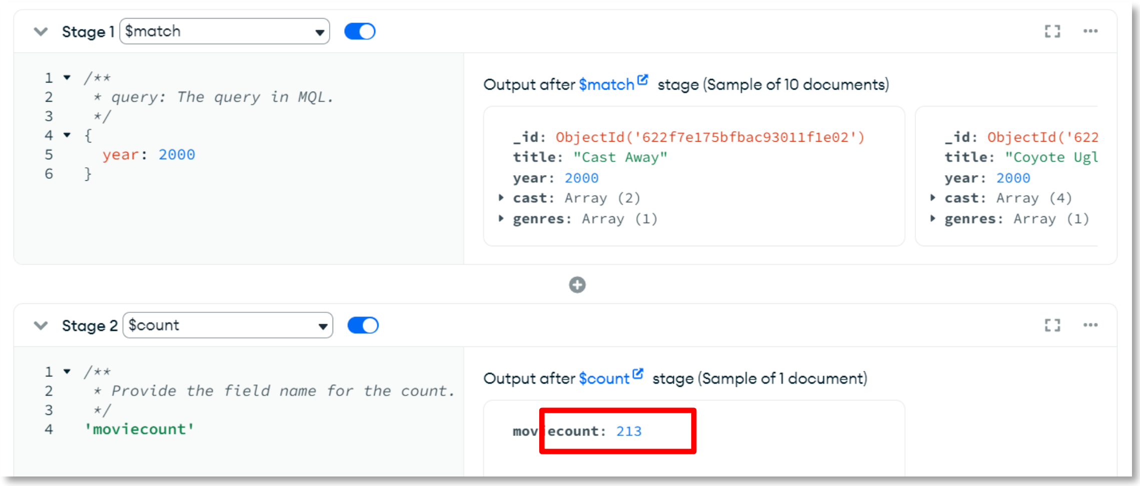
3DM – MongoDB Exercise 2

You plan to participate in a quiz about movies. To prepare yourself in the best possible way, you implement a small application that generates random quizzes from a database.

a) Open Compass and create a new collection named «movie» in your MongoDB on Atlas.
b) Import the movie dataset (available from Moodle) with Compass.
c) Follow the instructions in the tutorial «MongoDB Java Project Setup» to set up a Java project. Use **`moviequiz`** as artifact Id.
d) Open the collection in your Java application. Print the document count to the console and verify that you get a total count of 28795 documents.
e) Open «Aggregations» in Compass. Assemble a query to count the number of movies released in the year 2000. You should get 213.



```
/**
 * query: The query in MQL.
 */
{
  year: 2000
}
```

Output after $match stage (Sample of 10 documents)

```
_id: ObjectId('622f7e175bfbac93011f1e02')
title: "Cast Away"
year: 2000
cast: Array (2)
genres: Array (1)
```

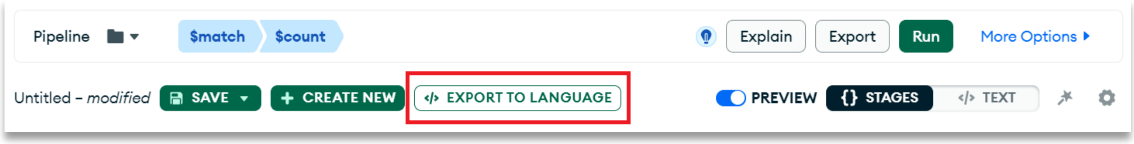
```
_id: ObjectId('622
title: "Coyote Ugl
year: 2000
cast: Array (4)
genres: Array (1)
```

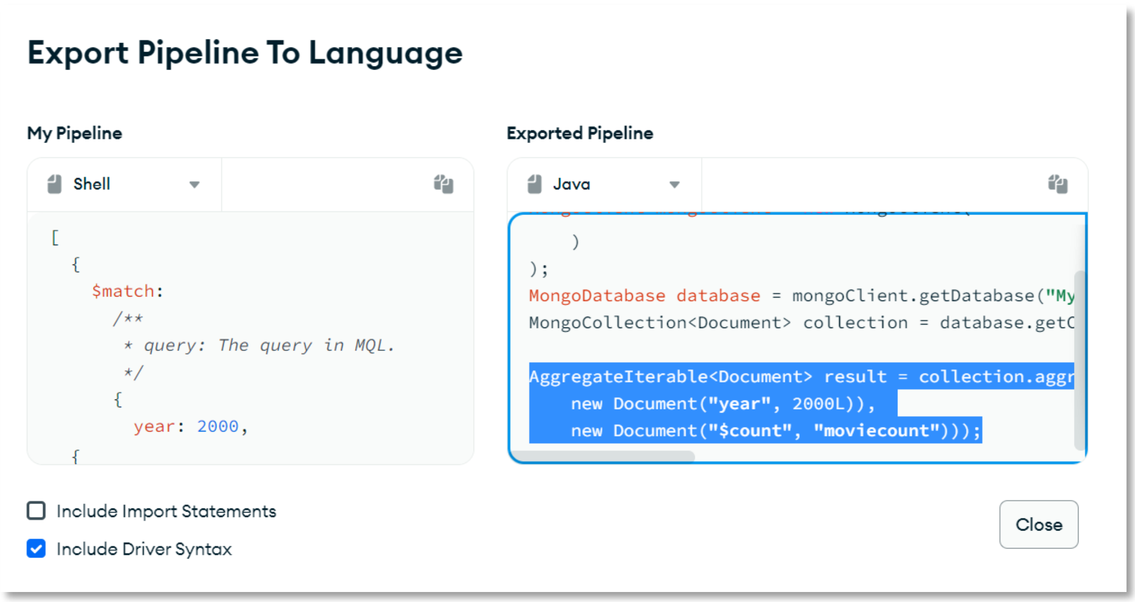
```
/**
 * Provide the field name for the count.
 */
'moviecount'
```

Output after $count stage (Sample of 1 document)

```
mov ecount: 213
```

f) Export and copy the pipeline to Java:





Export Pipeline To Language

My Pipeline

Shell

```
[
  {
    $match:
      /**
       * query: The query in MQL.
       */
      {
        year: 2000,
  {
```

Exported Pipeline

Java

```
    )
);
MongoDatabase database = mongoClient.getDatabase("My
MongoCollection<Document> collection = database.getC

AggregateIterable<Document> result = collection.aggr
    new Document("year", 2000L)),
    new Document("$count", "moviecount")));
```

☐ Include Import Statements

☑ Include Driver Syntax

g) Paste the aggregate query below the print of the document count. The code below shows how the movie count for 2000 can be printed to the console.

```
AggregateIterable<Document> result= movieCol.aggregate(
  Arrays.asList(new Document("$match",
    new Document("year", 2000L)),
    new Document("$count", "moviecount")));

System.out.println("Count for 2000: "+ result.first().get("moviecount"));
```

Explanation:

- Aggregate always returns an AggregateIterable of Documents. Like FindIterable used in Exercise 1, it is possible to iterate over it with a for-each loop.
- Because we know that this query always returns a single document containing the count, we can use first() to extract this document.

Run the application. It should print 213 to the terminal.

```
Found 28795 movies
Count for 2000: 213
```

h) Let the user enter a year and change the query to use this year. **Tip**: in the query, replace «2000L» with the variable containing the user input. Example terminal session:

```
Total count: 28795
Select a year
> 1950
Count for 1950: 443
```

i) Create a query that lists all distinct genres of the selected year. **Tip**: You need $match to select the year, $unwind to flatten the genre and $group to get the distinct genres. The example below shows how the result can be printed by transforming the result into an ArrayList.

```
AggregateIterable<Document> genresOfYear = movieCol.aggregate(...);

ArrayList<Document> genresList= genresOfYear.into(new ArrayList<Document>());
for (int i = 0; i < genresList.size(); i++) {
  System.out.println((i + 1) + ": " + genresList.get(i).get("_id"));
}
```

Why using an ArrayList? The types «AggregateIterable » and «FindIterable» are both a kind of list, but they do not allow you to access individual objects. You can only process them elementwise with a for-each loop. But both iterables can be transformed into an ArrayList when access to individual elements is required.

Example output for 1910:

```
Select a year
> 1910
Count for 1910: 26
1: Fantasy
2: Romance
3: Western
4: Documentary
5: Short
6: Drama
7: Comedy
```

Note: The order of genres is not fixed and changes on every run.

j) Let the user select a genre by its number and print the name of the selected genre.

```
Select a year
> 1912
Count for 1912: 44
1: Drama
2: Western
3: Horror
4: Comedy
5: Thriller
6: War
7: Adventure
8: Romance
Select a genre (1-8)
> 4
Selected Comedy
```

k) Now let's implement the quiz: Set up two new queries:

- The first query randomly picks a movie of the selected year and genre. **Tip**: Use $sample for random selection.
- The second query randomly picks two other movies that are 5-10 years older but from the same genre. **Tip**: Use $gt (year-10) and $lt (year-5).

Print the names of all three movies in random order and let the user guess which one is not from the selected year. **Tip** for random ordering: Save alle three movies to the same ArrayList and use Collections.shuffle(..). Check the answer and show the correct solution.

Example on next page..

```
Select a year
> 1920
Count for 1920: 129
1: Thriller
2: Adventure
3: Historical
4: Romance
5: Documentary
6: Western
7: Drama
8: Comedy
9: Horror
10: Mystery
11: Crime
12: Short
Select a genre (1-12)
> 6
Selected Western
1: The Broncho Kid
2: In the Secret Service
3: True Western Hearts
Which movie was released in 1920?
> 2
Wrong!
The Broncho Kid was released in 1920
In the Secret Service was released in 1913
True Western Hearts was released in 1914
```

l) **You made it! Submit** the main class (App.java) of your project on Moodle.

m) **Optional extra task**: Extend the query for the distinct genres to retrieve the total number of movies of each genre. Extend the printing to show this number (in brackets). **Tip**: You need $count as accumulator in $group.

```
Select a year
> 1910
Count for 1910: 26
1: Western (2)
2: Fantasy (1)
3: Drama (12)
4: Comedy (3)
5: Short (2)
6: Documentary (1)
7: Romance (1)
...
```