Class 7:
Aggregation pipeline

Aggregation: An aggregation pipeline is a sequence of stages that process data from a MongoDB collection. Each stage performs a specific operation on the data, such as filtering, transforming, or aggregating.

The output of one stage becomes the input for the next stage, allowing you to chain multiple operations together to achieve complex data processing tasks.

● Mongodb Aggregation Pipeline consist of stages and each stage transforms the document. It is a multi-stage pipeline and in each state and the documents are taken as input to produce the resultant set of documents.

In the next stage (ID available) the resultant documents are taken as input to produce output, this process continues till the last stage.

● **The basic pipeline stages are defined below:**
1. filters that will operate like queries.

2. the document transformation that modifies the resultant document.

3. provide pipeline provides tools for grouping and sorting documents.

● Aggregation pipeline can also be used in sharded collection.

● Execute Aggregation Pipeline and its operations (pipeline must contain $match, $group, $sort, $project

● $skip etc. students encourage to execute several queries to demonstrate various aggregation operators)

Explanation:

Explanation of Operators:

- `$match` : Filters documents based on a condition.
- `$group` : Groups documents by a field and performs aggregations like `$avg` (average) and `$sum` (sum).
- `$sort` : Sorts documents in a specified order (ascending or descending).
- `$project` : Selects specific fields to include or exclude in the output documents.
- `$skip` : Skips a certain number of documents from the beginning of the results.
- `$limit` : Limits the number of documents returned.
- `$unwind` : Deconstructs an array into separate documents for each element.

These queries demonstrate various aggregation operations using the `students6` collection. Feel free to experiment with different conditions and operators to explore the power of aggregation pipelines in MongoDB.

Projection Operator :

projection operators allow users to control queries and results. Query operators help to filter data based on specific conditions.

E.g., $eq,$and,$exists, etc. Projection operators in MongoDB control which fields should be shown in the results. E.g., $project, $slice, $concat, etc.

1. Find students with age greater than 23, sorted by age in descending order, and only return name and age

```
db.students6.aggregate([
  { $match: { age: { $gt: 23 } } }, // Filter students older than 23
  { $sort: { age: -1 } }, // Sort by age descending
  { $project: { _id: 0, name: 1, age: 1 } } // Project only name and
])
```

Ans. Find students with age greater than 23, sorted by age in descending order, and only return name and age.

```
db> db.students6.aggregate([
...   { $match: { age: { $gt: 23 } } }, // Filter students older than 23
...   { $sort: { age: -1 } }, // Sort by age descending
...   { $project: { _id: 0, name: 1, age: 1 } } // Project only name and age
... ])
[ { name: 'Charlie', age: 28 }, { name: 'Alice', age: 25 } ]
db>
```

$match- filters students' age older than 23.

$sort- sorts the age by descending order .

$project- projects only name and age.

2. Group students by major, calculate average age and total number of students in each major:

```
db> db.students6.aggregate([
...   { $group: { _id: "$major", averageAge: { $avg: "$age" }, totalStudents: { $sum: 1 } } }
... ])
[
  { _id: 'Mathematics', averageAge: 22, totalStudents: 1 },
  { _id: 'English', averageAge: 28, totalStudents: 1 },
  { _id: 'Computer Science', averageAge: 22.5, totalStudents: 2 },
  { _id: 'Biology', averageAge: 23, totalStudents: 1 }
]
```

3. Find students with an average score (from scores array) above 85 and skip the first document

```
db.students6.aggregate([
  {
    $project: {
      _id: 0,
      name: 1,
      averageScore: { $avg: "$scores" }
    }
  },
  { $match: { averageScore: { $gt: 85 } } },
  { $skip: 1 } // Skip the first document
])
```

Ans: Find students with an average score (from scores array) above 85 and skip the first document