AGGREGATION OPERATORS:

Aggregate operators in MongoDB are used to perform data processing and transformation on data in a pipeline fashion. They are used in the **`$aggregate`** stage of a MongoDB query to process and transform data in a collection.

Types of Aggregate Operators

- `$sum`: Calculates the sum of a set of values.
- `$avg`: Calculates the average of a set of values.
- `$min`: Returns the minimum value in a set of values.
- `$max`: Returns the maximum value in a set of values.
- `$push`: Adds an element to an array.
- `$addToSet`: Adds an element to an array, but only if it doesn't already exist.

AVERAGE GPA OF ALL STUDENTS:

```
JavaScript


db.students.aggregate([
  { $group: { _id: null, averageGPA: { $avg: "$gpa" } } }
]);
```

ANSWER:

```
[ { _id: null, averageGPA: 2.98556 } ]
db>
```

- `$group`: Groups all documents together.
  - `_id: null`: Sets the group identifier to null (optional, as there's only one group in this case).
  - `averageGPA`: Calculates the average value of the "gpa" field using the `$avg` operator.

MINIMUM AND MAXIMUM AGE:

```
db> db.students.aggregate([
...   { $group: { _id: null, minAge: { $min: "$age" }, maxAge: { $max: "$age" } } }
... ]);
```

ANSWER:

```
[ { _id: null, minAge: 18, maxAge: 25 } ]
```

Similar to the previous example, it uses **$group** to group all documents.

minAge: Uses the $min operator to find the minimum value in the "age" field.

maxAge: Uses the $max operator to find the maximum value in the "age" field.

AVERAGE GPA OF ALL HOME CITIES:

```
db> db.students.aggregate([
...   { $group: { _id: "$home_city", averageGPA: { $avg: "$gpa" } } }
... ]);
[
  { _id: 'City 8', averageGPA: 3.11741935483871 },
  { _id: 'City 7', averageGPA: 2.8479310344827 59 },
  { _id: 'City 10', averageGPA: 2.935227272727273 },
  { _id: 'City 9', averageGPA: 3.1174358974358976 },
  { _id: 'City 2', averageGPA: 3.01969696969697 },
  { _id: 'City 3', averageGPA: 3.0100000000000002 },
  { _id: 'City 6', averageGPA: 2.896944444444448 },
  { _id: null, averageGPA: 2.9784313725490197 },
  { _id: 'City 4', averageGPA: 2.8251851851851852 },
  { _id: 'City 1', averageGPA: 3.003823529411765 },
  { _id: 'City 5', averageGPA: 3.0607499999999996 }
]
db>
```

PUSHING ALL COURSES INTO SINGLE ARRAY:

```
db.students.aggregate([
  { $project: { _id: 0, allCourses: { $push: "$courses" } } }
]);
```

- **$project:** Transforms the input documents.

- **`_id: 0`:** Excludes the `_id` field from the output documents.
- **`allCourses`:** Uses the `$push` operator to create an array. It pushes all elements from the "courses" field of each student document into the **`allCourses`** array.

  This will return a list of documents, each with an **`allCourses`** array containing all unique courses offered (assuming courses might be duplicated across students).

BUT:

```
db> db.students.aggregate([
...   { $project: { _id: 0, allCourses: { $push: "$courses" } } }
... ]);
MongoServerError[Location31325]: Invalid $project :: caused by :: Unknown expression $push
db>
```

$addToset:

```
db.candidates.aggregate([

 { $unwind: "$courses" },

 { $group: { _id: null, uniqueCourses: { $addToSet: "$courses" } } }

]);
```

```
db> db.candidates.aggregate([
...   { $unwind: "$courses" }, // Deconstruct courses array
...   { $group: { _id: null, uniqueCourses: { $addToSet: "$courses" } } }
que courses
... ]);
[
  {
    _id: null,
    uniqueCourses: [
      'Sociology',
      'Literature',
      'Ecology',
      'Physics',
      'Mathematics',
      'Marine Science',
      'Artificial Intelligence',
      'Art History',
      'Creative Writing',
      'Robotics',
      'Environmental Science',
      'Biology',
      'Statistics',
      'Music History',
      'Philosophy',
      'Film Studies',
      'Engineering',
      'Computer Science',
      'English',
      'Psychology',
      'Chemistry',
      'Political Science',
```

AGGREGATION PIPELINE:

An aggregation pipeline in MongoDB is a series of data processing stages that transform and aggregate data from a collection. Each stage in the pipeline processes the data and passes the output to the next stage.

Here are some common stages that can be used in an aggregation pipeline:

1. **$match**: Filters the data based on a condition.
2. **$project**: Transforms the data by adding or modifying fields.
3. **$group**: Groups the data based on one or more fields and performs aggregation operations.
4. **$sort**: Sorts the data in ascending or descending order.
5. **$limit**: Limits the number of documents in the output.
6. **$lookup**: Performs a left outer join with another collection.
7. **$unwind**: Deconstructs an array field into separate documents.

8. **$out**: Writes the output to a new collection.

Each stage in the pipeline processes the data and passes the output to the next stage, allowing for complex data transformations and aggregations to be performed.

Examples:

Example 1:

Filtering and Grouping:

Question: What is the total sales amount for each region in 2022?

```
db.sales.aggregate([
 {
  $match: { date: { $gte: ISODate("2022-01-01T00:00:00.000Z") } }
 },
 {
  $group: {
   _id: "$region",
   totalSales: { $sum: "$amount" }
  }
 }
])
```

Example 2:

Sorting and Limiting

Question: What are the top 5 sales amounts for Product A?

```
db.sales.aggregate([
 {
  $match: { product: "Product A" }
 },
 {
  $sort: { amount: -1 }
 },
 {
  $limit: 5
 }
])
```

Example 3:

Joining Collections

Question: What are the product names and sales amounts for each sale?

```
db.sales.aggregate([
 {
  $lookup: {
   from: "products",
   localField: "product",
   foreignField: "_id",
```

```
    as: "productDetails"
   }
  },
  {
   $unwind: "$productDetails"
  },
  {
   $project: {
    _id: 0,
    product: "$productDetails.name",
    amount: 1
   }
  }
])
```