Experiment 4:

Projection Operators

Create and demonstrate how projection operators ($, $elematch and $slice) would be used in the MondoDB.

To demonstrate the use of projection operators (`$`, `$elemMatch`, and `$slice`) in MongoDB, let's create a `Products` collection. We'll insert documents that include arrays, which will allow us to showcase these operators effectively.

Create the `Products` Collection and Insert Documents

```
test> use retailDB
retailDB> db.Products.insertMany([{
    name: "Laptop",
    brand: "BrandA",
    features: [
      { name: "Processor", value: "Intel i7" },
      { name: "RAM", value: "16GB" },
      { name: "Storage", value: "512GB SSD" }
    ],
    reviews: [
      { user: "Alice", rating: 5, comment: "Excellent!" },
      { user: "Bob", rating: 4, comment: "Very good" },
      { user: "Charlie", rating: 3, comment: "Average" }
    ]
  },
  {
    name: "Smartphone",
    brand: "BrandB",
    features: [
      { name: "Processor", value: "Snapdragon 888" },
      { name: "RAM", value: "8GB" },
      { name: "Storage", value: "256GB" }
    ],
    reviews: [
      { user: "Dave", rating: 4, comment: "Good phone" },
      { user: "Eve", rating: 2, comment: "Not satisfied" }
    ]
  }
])
```

Use Projection Operators

1. The `$` Projection Operator

The `$` operator is used to project the first matching element from an array of embedded documents.

Example: Find the product named “Laptop” and project the review from the user “Alice”.

```
retailDB> db.Products.find(
  { name: "Laptop", "reviews.user": "Alice" },
  { "reviews.$": 1 }
).pretty()
```

Output

```
{  "_id": ObjectId("..."),
  "reviews": [{ "user": "Alice", "rating": 5, "comment": "Excellent!" }]}
```

2. The `$elemMatch` Projection Operator

The `$elemMatch` operator is used to project the first matching element from an array based on specified criteria.

Example: Find the product named “Laptop” and project the review where the rating is greater than 4.

```
retailDB> db.Products.find(
  { name: "Laptop" },
  { reviews: { $elemMatch: { rating: { $gt: 4 } } } }
).pretty()
```

Output:

```
{
  "_id": ObjectId("..."),
  "reviews": [
    { "user": "Alice", "rating": 5, "comment": "Excellent!" }
  ]
}
```

3. The `$slice` Projection Operator

The `$slice` operator is used to include a subset of the array field.

Example: Find the product named “Smartphone” and project the first review.

```
retailDB> db.Products.find(
  { name: "Smartphone" },
  { reviews: { $slice: 1 } }
).pretty()
```

Output:

```
{
  "_id": ObjectId("..."),
  "reviews": [
    { "user": "Dave", "rating": 4, "comment": "Good phone" }
  ]
}
```

Additional Example with Multiple Projection Operators

Example: Find the product named “Laptop” and project the `name`, the first two features, and the review with the highest rating.

```
retailDB> db.Products.find(
  { name: "Laptop" },
  {
    name: 1,
    features: { $slice: 2 },
    reviews: { $elemMatch: { rating: 5 } }
  }
).pretty()
```

Output:

```
{
  "_id": ObjectId("..."),
  "name": "Laptop",
  "features": [
    { "name": "Processor", "value": "Intel i7" },
    { "name": "RAM", "value": "16GB" } ],"reviews": [ { "user": "Alice", "rating": 5, "comment":
"Excellent!" }]}
```

Using projection operators in MongoDB, you can fine-tune the data returned by your queries:

- The `$` operator is useful for projecting the first matching element from an array.
- The `$elemMatch` operator allows you to project the first array element that matches specified criteria.
- The `$slice` operator lets you project a subset of an array, such as the first `n` elements or a specific range.

Experiment 5:

Execute Aggregation operations ($avg, $min,$max, $push, $addToSet etc.). students encourage to execute several queries to demonstrate various aggregation operators)

Aggregation operations

Execute Aggregation operations (*avg*,min,*max*,push, $addToSet etc.). students encourage to execute several queries to demonstrate various aggregation operators)

To demonstrate aggregation operations such as `$avg`, `$min`, `$max`, `$push`, and `$addToSet` in MongoDB, we will use a `Sales` collection. This collection will contain documents representing sales transactions.

Create the `Sales` Collection and Insert Documents

First, we'll create the `Sales` collection and insert sample documents.

```
test> use salesDB

salesDB> db.Sales.insertMany([
  { date: new Date("2024-01-01"), product: "Laptop", price: 1200, quantity: 1, customer: "Amar" },
  { date: new Date("2024-01-02"), product: "Laptop", price: 1200, quantity: 2, customer: "Babu" },
  { date: new Date("2024-01-03"), product: "Mouse", price: 25, quantity: 5, customer: "Chandra" },
  { date: new Date("2024-01-04"), product: "Keyboard", price: 45, quantity: 3, customer: "Amar" },
  { date: new Date("2024-01-05"), product: "Monitor", price: 300, quantity: 1, customer: "Babu" },
  { date: new Date("2024-01-06"), product: "Laptop", price: 1200, quantity: 1, customer: "Deva" }
])
```

Execute Aggregation Operations

1. $avg (Average)

Calculate the average price of each product.

```
salesDB> db.Sales.aggregate([
  {
    $group: {
      _id: "$product",
      averagePrice: { $avg: "$price" }
    }
  }
]).pretty()
```

Output:

```
[
  { "_id": "Laptop", "averagePrice": 1200 },
  { "_id": "Mouse", "averagePrice": 25 },
  { "_id": "Keyboard", "averagePrice": 45 },
  { "_id": "Monitor", "averagePrice": 300 }
]
```

2. `$min` (Minimum)

Find the minimum price of each product.

```
salesDB> db.Sales.aggregate([
  {
    $group: {
      _id: "$product",
      minPrice: { $min: "$price" }
    }
  }
]).pretty()
```

Output:

```
[
  { "_id": "Laptop", "minPrice": 1200 },
  { "_id": "Mouse", "minPrice": 25 },
  { "_id": "Keyboard", "minPrice": 45 },
  { "_id": "Monitor", "minPrice": 300 }
]
```

3. `$max` (Maximum)

Find the maximum price of each product.

```
salesDB> db.Sales.aggregate([
  {
    $group: {
      _id: "$product",
      maxPrice: { $max: "$price" }
    }
  }
]).pretty()
```

Output:

```
[
  { "_id": "Laptop", "maxPrice": 1200 },
  { "_id": "Mouse", "maxPrice": 25 },
  { "_id": "Keyboard", "maxPrice": 45 },
  { "_id": "Monitor", "maxPrice": 300 }
]
```

4. `$push` (Push Values to an Array)

Group sales by customer and push each purchased product into an array.

```
salesDB> db.Sales.aggregate([
  {
    $group: {
      _id: "$customer",
      products: { $push: "$product" }
    }
  }
]).pretty()
```

Output:

```
[
  { "_id": "Amar", "products": ["Laptop", "Keyboard"] },
  { "_id": "Babu", "products": ["Laptop", "Monitor"] },
  { "_id": "Chandra", "products": ["Mouse"] },
  { "_id": "Deva", "products": ["Laptop"] }
]
```

5. `$addToSet` (Add Unique Values to an Array)

Group sales by customer and add each unique purchased product to an array.

```
salesDB> db.Sales.aggregate([
  {
    $group: {
      _id: "$customer",
      uniqueProducts: { $addToSet: "$product" }
    }
  }
]).pretty()
```

Output:

```
[
  { "_id": "Amar", "uniqueProducts": ["Laptop", "Keyboard"] },
  { "_id": "Babu", "uniqueProducts": ["Laptop", "Monitor"] },
  { "_id": "Chandra", "uniqueProducts": ["Mouse"] },
  { "_id": "Deva", "uniqueProducts": ["Laptop"] }
]
```

Combining Aggregation Operations

Let's combine several aggregation operations to get a comprehensive report.

Example: Calculate the total quantity and total sales amount for each product, and list all customers who purchased each product.

```
salesDB> db.Sales.aggregate([
  {
    $group: {
      _id: "$product",
      totalQuantity: { $sum: "$quantity" },
      totalSales: { $sum: { $multiply: ["$price", "$quantity"] } },
      customers: { $addToSet: "$customer" }
    }
  }
]).pretty()
```

Output:

```
[
  {
    "_id": "Laptop",
    "totalQuantity": 4,
    "totalSales": 4800,
    "customers": ["Amar", "Babu", "Deva"]
  },
  {
    "_id": "Mouse",
    "totalQuantity": 5,
    "totalSales": 125,
    "customers": ["Chandra"]
  },
  {
    "_id": "Keyboard",
    "totalQuantity": 3,
    "totalSales": 135,
    "customers": ["Amar"]
  },
  {
    "_id": "Monitor",
    "totalQuantity": 1,
    "totalSales": 300,
    "customers": ["Babu"]
  }
]
```

By using aggregation operations such as `$avg`, `$min`, `$max`, `$push`, and `$addToSet`, you can perform complex data analysis and transformations on MongoDB collections. These operations enable you to calculate averages, find minimum and maximum values, push values into arrays, and create sets of unique values. The examples provided show how to use these operators to analyze a `Sales` collection.