Experiment 4:

Demonstrate how projection operators ($, $elematch and $slice) would be used in the MongoDB.

Solution:

use 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. $ Projection Operator: Find the product named “Laptop” and project the review from the user “Alice”.

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

OUTPUT:

```
[
  {
    _id: ObjectId('666c2f237d3bfa1feacdce05'),
    reviews: [ { user: 'Alice', rating: 5, comment: 'Excellent!' } ]
  }
]
```

2. $elemMatch Projection Operator: Find the product named “Laptop” and project the review where the rating is greater than 4.

```
db.Products.find(
  { name: "Laptop" },
```

```
  { reviews: { $elemMatch: { rating: { $gt: 4 } } } }
).pretty()
```

OUTPUT:

```
[
  {
    _id: ObjectId('666c2f237d3bfa1feacdce05'),
    reviews: [ { user: 'Alice', rating: 5, comment: 'Excellent!' } ]
  }
]
```

3. $slice Projection Operator: Find the product named “Smartphone” and project the first review.

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

OUTPUT:

```
[
  {
    _id: ObjectId('666c2f237d3bfa1feacdce06'),
    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' } ]
  }
]
```

Experiment 5:

Execute Aggregation operations ($avg, $min,$max, $push, $addToSet etc.).

(students encourage to execute several queries to demonstrate various aggregation operators)

Solution:

```
use 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.

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

OUTPUT:

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

2. $min (Minimum): Find the minimum price of each product.

```
db.Sales.aggregate([
 {
  $group: {
   _id: "$product",
   minPrice: { $min: "$price" }
  }
 }
```

]).pretty()

OUTPUT:

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

3. $max (Maximum): Find the maximum price of each product.

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

OUTPUT:

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

4. $push (Push Values to an Array): Group sales by customer and push each purchased product into an array.

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

OUTPUT:

```
[
 { _id: 'Babu', products: [ 'Laptop', 'Monitor' ] },
 { _id: 'Amar', products: [ 'Laptop', 'Keyboard' ] },
 { _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.

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

OUTPUT:

```
[
```

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

Combining Aggregation Operations:

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

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

OUTPUT:

```
[
  {
    _id: 'Mouse',
    totalQuantity: 5,
    totalSales: 125,
    customers: [ 'Chandra' ]
```

```
  },
  {
    _id: 'Keyboard',
    totalQuantity: 3,
    totalSales: 135,
    customers: [ 'Amar' ]
  },
  {
    _id: 'Monitor',
    totalQuantity: 1,
    totalSales: 300,
    customers: [ 'Babu' ]
  },
  {
    _id: 'Laptop',
    totalQuantity: 4,
    totalSales: 4800,
    customers: [ 'Amar', 'Babu', 'Deva' ]
  }
]
```