Aggregation for Product Details

```
use("Ecommerce")

db.products.aggregate([
    {
        $lookup: {
            from: 'brands',
            localField: 'brandID',
            foreignField: '_id',
            as: 'brandDetails'
        }
    },
    {
        $lookup: {
            from: 'categories',
            localField: 'categoryID',
            foreignField: '_id',
            as: 'categoriesDetails'
        }
    }
])
```

Comprehensive Product Data Retrieval

This MongoDB aggregation pipeline for the `products` collection in the `Ecommerce` database enhances product documents by performing the following operations

```
db.products.aggregate([
    {
        $lookup: {
            from: 'brands',
            localField: 'brandID',
            foreignField: '_id',
            as: 'brandDetails'
        }
    },
    {
        $lookup: {
            from: 'categories',
            localField: 'categoryID',
            foreignField: '_id',
            as: 'categoryDetails'
        }
    },
```

```
        {
                $project: {
                        _id: 0,
                        title: 1,
                        shortDes: 1,
                        brandName: { $first: "$brandDetails.brandName" },
                        categoryName: { $first:
"$categoryDetails.categoryName" },
                        price: { $toDouble: "$price" },
                        discount: 1,
                        discountP: { $toDouble: "$discountPrice" },
                        image: 1,
                        star: 1,
                        stock: 1,
                        remark: 1,
                        categoryID: 1,
                        brandID: 1,
                }
        },
        {   $addFields: {FinalPrice:
                        {$subtract:[
                                                {$toDouble:"$price" },{
$toDouble: "$discountP" }
                                        ]
                        }
                        }
        },
        {
                $addFields: {Random: { $subtract: [4500, 454] }}
        }
])
```

- **Joins with `brands` and `categories` Collections:** Uses the `$lookup` operator to add detailed brand and category information to each product.
- **Projects Specific Fields:** The `$project` stage selects specific fields to include in the output, converts prices to double, and finds brand and category names.
- **Calculates Final Price:** Adds a `FinalPrice` field by subtracting the `discount price` from the `original price`.
- **Adds a Random Calculation Field:** Includes a `Random` field for demonstration purposes, showcasing an arbitrary subtraction operation.

Introducing $facet

```
db.products.aggregate([
{ $facet:{
                "total":[{$count:"total"}],
                "data":[]
```

```
        }}
])
```

- **$facet:** Allows for the execution of multiple stages in parallel and outputs the results in separate fields.
- **"total":** A facet that counts the total number of documents in the `products` collection.
- **"data":** An empty facet, demonstrating the capability to include other operations or projections as needed.

Using Facets for Total Count, Brand-Specific Data, and Price Filtering

```
db.products.aggregate([
{ $facet:{
        "total":[{$count:"total"}],
        "brand":[
                        {$match:{"brandID" :
ObjectId("64f8751a502e1b80556da142")}},
                        {$limit:2}
                        ],
        "price":[{$match:{"price":{$gt:"25000"}}}]
        }}
])
```

- **$facet:** Allows multiple aggregations to be executed in parallel, with each facet producing its own output.
- **"total":** Counts the total number of documents in the `products` collection.
- **"brand":** Filters documents to those with a specific `brandID` and limits the result to 2 documents.
- **"price":** Filters documents where the `price` field is greater than 25000.

Concatenation, String Splitting, Uppercase Conversion & Calculation

```
db.products.aggregate([
    {
        $addFields: {
            usingConcat: {
                $concat: ["$title", " ", "$price", " ", "$remark"]
            }
        }
    },
    {
        $addFields: {
            usingSplit: {
```

```
                $split: ["$title", " "]
            }
        }
    },
    {
        $addFields: {
            usingUpper: {
                $toUpper: "$title"
            }
        }
    },
    {
        $addFields: {
            FinalPrice: {
                $subtract: [
                    { $toDouble: "$price" },
                    { $toDouble: "$discountPrice" }
                ]
            }
        }
    }
])
```

Concatenation with $concat:

- **usingConcat:** Combines the `title`, `price`, and `remark` fields into a single string, separated by spaces.

String Splitting with $split:

- **usingSplit:** Splits the `title` field into an array of words using space as the delimiter.

Uppercase Conversion with $toUpper:

- **usingUpper:** Converts the `title` field to uppercase.

Final Price Calculation:

- **FinalPrice:** Computes the final price by subtracting the `discountPrice` (converted to a double) from the `price` (also converted to a double).

Extracting Date and Time Components

```
db.products.aggregate([
    { $addFields: { dateOfYear: { $dayOfYear: "$createdAt" } } },
    { $addFields: { MonthOfYear: { $dayOfMonth: "$createdAt" } } },
    { $addFields: { WeekOfYear: { $dayOfWeek: "$createdAt" } } },
```

```
    { $addFields: { Year: { $year: "$createdAt" } } },
    { $addFields: { week: { $week: "$createdAt" } } },
    { $addFields: { hour: { $hour: "$createdAt" } } },
    { $addFields: { minute: { $minute: "$createdAt" } } },
    { $addFields: { second: { $second: "$createdAt" } } },
    { $addFields: { millisce: { $millisecond: "$createdAt" } } }
])
```

Explanation:

- **dateOfYear:** Extracts the day of the year (263rd day of the year).
- **MonthOfYear:** Extracts the day of the month (20th day of the month).
- **WeekOfYear:** Extracts the day of the week (4th day, usually Wednesday).
- **Year:** Extracts the year (2023).
- **week:** Extracts the week of the year (38th week).
- **hour:** Extracts the hour (17, or 5 PM).
- **minute:** Extracts the minute (17th minute).
- **second:** Extracts the second (56th second).
- **millisce:** Extracts the millisecond (893rd millisecond).

This pipeline provides a detailed breakdown of the `createdAt` timestamp, which can be used for various analysis and reporting purposes within the `products` collection

And The Result:

```
{
    "_id": ObjectId("60a6c5f9d5f9b54e1f4d2e2c"),
    "title": "Product 1",
    "createdAt": ISODate("2023-09-20T17:17:56.893Z"),
    "dateOfYear": NumberInt(263),
    "MonthOfYear": NumberInt(20),
    "WeekOfYear": NumberInt(4),
    "Year": NumberInt(2023),
    "week": NumberInt(38),
    "hour": NumberInt(17),
    "minute": NumberInt(17),
    "second": NumberInt(56),
    "millisce": NumberInt(893)
}
```

Categorizing Employee Salaries

```
db.employees.aggregate([
    {
        $project: {
```

```
            _id: 0,
            salary: 1,
            salaryMargin: {
                $cond: {
                    if: { $lt: [ { $toDouble: "$salary" }, 90000 ] },
                    then: "LOW Salary",
                    else: "Standard Salary"
                }
            }
        }
    }
])
```

Explanation:

- **$project:** Specifies the fields to include in the output documents.
  - **_id: 0:** Excludes the `_id` field from the output.
  - **salary:** Includes the `salary` field in the output.
  - **salaryMargin:** Adds a new field that categorizes the salary based on the specified condition.
    - **$cond:** A conditional operator that evaluates an expression and returns one of two specified values.
      - **if:** The condition to evaluate. In this case, it checks if the `salary` (converted to a double) is less than 90000.
      - **then:** The value to return if the condition is true ("LOW Salary").
      - **else:** The value to return if the condition is false ("Standard Salary").

And The Result:

```
[
    { "salary": NumberInt(85000), "salaryMargin": "LOW Salary" },
    { "salary": "105000", "salaryMargin": "Standard Salary" },
    { "salary": NumberInt(90000), "salaryMargin": "Standard Salary" },
    { "salary": NumberInt(85000), "salaryMargin": "LOW Salary" },
    { "salary": NumberInt(105000), "salaryMargin": "Standard Salary" },
    { "salary": "90000", "salaryMargin": "Standard Salary" },
    { "salary": "101000", "salaryMargin": "Standard Salary" },
    { "salary": "65000", "salaryMargin": "LOW Salary" },
    { "salary": "68000", "salaryMargin": "LOW Salary" },
    { "salary": "59000", "salaryMargin": "LOW Salary" },
    { "salary": "75000", "salaryMargin": "LOW Salary" },
    { "salary": "72000", "salaryMargin": "LOW Salary" },
    { "salary": "55000", "salaryMargin": "LOW Salary" }
]
```

Categorizing Employee Salaries with Multiple Ranges

```
db.employees.aggregate([
    {
        $project: {
            _id: 0,
            name: 1,
            salary: 1,
            salaryMargin: {
                $switch: {
                    branches: [
                        {
                            case: { $lte: [{ $toDouble: "$salary" }, 60000]
},
                            then: "LOW Salary"
                        },
                        {
                            case: { $and: [
                                { $gt: [{ $toDouble: "$salary" }, 60000] },
                                { $lte: [{ $toDouble: "$salary" }, 70000] }
                            ] },
                            then: "MEDIUM Salary"
                        },
                        {
                            case: { $and: [
                                { $gt: [{ $toDouble: "$salary" }, 70000] },
                                { $lte: [{ $toDouble: "$salary" }, 80000] }
                            ] },
                            then: "HIGH Salary"
                        }
                    ],
                    default: "VERY HIGH Salary"
                }
            }
        }
    }
])
```

And The Result:

```
[
    { "name": "Alice", "salary": 55000, "salaryMargin": "LOW Salary" },
    { "name": "Bob", "salary": 65000, "salaryMargin": "MEDIUM Salary" },
    { "name": "Charlie", "salary": 75000, "salaryMargin": "HIGH Salary" },
```

```
    { "name": "Dave", "salary": 95000, "salaryMargin": "VERY HIGH Salary" }
]
```