Aggregation

//\* $group

// The $group stage groups documents by specified fields and performs aggregation functions. it is like the reduce methods in JS

// when dealing with $group stage we need to pass $sign for our existing field not the one we are going to create

// syntax :

 {

  $group:

    {

      \_id: <expression>, // Group key

      <field1>: { <accumulator1> : <expression1> },

      ...

    }

 }

// https://www.mongodb.com/docs/v6.0/reference/operator/aggregation/group/#considerations

db.products.aggregate([

  { $match: { price: { $gt: 900 } } },

  {

    $group: {

      \_id: { sameCompany: "$company" },

      totalPrice: { $sum: "$price" },

    },

  },

]);

// let's use another accumulator operations

// $avg

// find the quantity = 5, group them with same quantity and find the average price

db.sales.aggregate([{ $match: { quantity: { $eq: 5 } } }]);

// both are same

db.sales.aggregate([

  { $match: { quantity: 5 } },

  {

    $group: {

      \_id: { quan: "$quantity" },

      avgPrice: { $avg: "$price" },

    },

  },

]);

//\* $sort

db.products.aggregate([

  { $match: { price: { $gt: 1200 } } },

  {

    $group: {

      \_id: "$category",

      totalPrice: { $sum: "$price" },

    },

  },

  { $sort: { totalPrice: 1 } },

]);

// $sort is like .sort() but you can even sort the values that you added in group. (Of course you can also sort before grouping or with any other values. But here you can even sort in ascending or descending based on number of products it has.

db.products.aggregate([

  { $match: { price: { $gt: 1200 } } },

  {

    $group: {

      \_id: "$category",

      totalPrice: { $sum: "$price" },

    },

  },

  { $sort: { totalPrice: -1 } },

]);

//\* $project

db.products.aggregate([

  {

    $project: {

      \_id: 0,

      price: 1,

      name: 1,

    },

  },

]);

// We can use the $project stage to create new fields by applying expressions or transformations to existing fields. For example, you could calculate the discounted price as a new field:

db.products.aggregate([

  { $match: { price: { $gt: 1000 } } },

  {

    $project: {

      \_id: 0,

      name: 1,

      originalPrice: "$price",

      disPrice: { $multiply: ["$price", 0.8] },

    },

  },

]);

// again we can add the sort here too

db.products.aggregate([

  { $match: { price: { $gt: 1000 } } },

  {

    $project: {

      \_id: 0,

      name: 1,

      originalPrice: "$price",

      disPrice: { $multiply: ["$price", 0.8] },

    },

  },

  { $sort: { disPrice: -1 } },

]);

//\* $push and $unwind

//? Find documents with a price greater than 1200, then group them by price and create an array of colors for each group.

//\* Before

//\* if price = 1250 =>  colors: [ '#000000', '#cc6600', '#663300' ],

//\* if price = 1250 =>  colors: [ '#fff000', '#ddddd', '#663300' ],

//? After,  I need a new document where

{

  price: 1250,

  allColors: ['#000000', '#cc6600', '#663300', '#fff000', '#ddddd', '#663300' ]

}

// code

db.products.aggregate([

  { $match: { price: { $gt: 1200 } } },

  {

    $group: {

      \_id: { priceGroup: "$price" },

      colors: { $push: "$colors" },

    },

  },

]);

//\* $unwind

db.products.aggregate([

  { $match: { price: { $gt: 1200 } } },

  { $unwind: "$colors" },

  {

    $group: {

      \_id: { priceGroup: "$price" },

      colors: { $push: "$colors" },

    },

  },

]);

//?  Before

{

  \_id: ObjectId("64c23601e32f4a51b19b9263"),

  name: 'Laptop Pro',

  company: '64c23350e32f4a51b19b9231',

  price: 1299,

  colors: [ '#333333', '#cccccc', '#00ff00' ],

  image: '/images/product-laptop.png',

  category: '64c2342de32f4a51b19b924e',

  isFeatured: true

},

//! $unwind: '$colors';

//? the $unwind stage deconstructs the "colors" array, creating multiple documents for each color within a product.

//?  After

{

  \_id: ObjectId("64c23601e32f4a51b19b9263"),

  name: 'Laptop Pro',

  company: '64c23350e32f4a51b19b9231',

  price: 1299,

  colors: '#333333',

  image: '/images/product-laptop.png',

  category: '64c2342de32f4a51b19b924e',

  isFeatured: true

},

{

  \_id: ObjectId("64c23601e32f4a51b19b9263"),

  name: 'Laptop Pro',

  company: '64c23350e32f4a51b19b9231',

  price: 1299,

  colors: '#cccccc',

  image: '/images/product-laptop.png',

  category: '64c2342de32f4a51b19b924e',

  isFeatured: true

},

// so now all the colors are in a string format, so $push will add them as an element in an array of colors

db.products.aggregate([

  { $match: { price: { $gt: 1200 } } },

  { $unwind: "$colors" },

  {

    $group: {

      \_id: null,

      totalCount: { $sum: 1 },

    },

  },

]);

db.products.aggregate([

  { $match: { price: { $gt: 1200 } } },

  { $unwind: "$colors" },

  {

    $group: {

      \_id: { priceGroup: "$price" },

      colors: { $push: "$colors" },

    },

  },

]);

//\* $addToSet

// still there is a problem and that is we are also getting the duplicates values so to remove it we will use the $addToSet

db.products.aggregate([

  { $match: { price: { $gt: 1200 } } },

  { $unwind: "$colors" },

  {

    $group: {

      \_id: { priceGroup: "$price" },

      colors: { $addToSet: "$colors" },

    },

  },

]);

//\* $size

// What If we want to count the number of unique colors for each price group

db.products.aggregate([

  { $match: { price: { $gt: 1200 } } },

  { $unwind: "$colors" },

  {

    $group: {

      \_id: { priceGroup: "$price" },

      colors: { $addToSet: "$colors" },

      colorLength: { $size: "$colors" },

    },

  },

]);

// we can't do this, bcz the $size operator is not allowed directly within the $group stage. Instead, you can use it in combination with other aggregation operators or in separate pipeline stages.

db.products.aggregate([

  { $match: { price: { $gt: 1200 } } },

  { $unwind: "$colors" },

  {

    $group: {

      \_id: { priceGroup: "$price" },

      allColors: { $addToSet: "$colors" },

    },

  },

  {

    $project: {

      \_id: 1,

      allColors: 1,

      colorLength: { $size: "$allColors" },

    },

  },

  { $limit: 1 },

]);

//! very Important in project stage we are only getting two fields and the name of the fields has to match with the fields names in group stage. ex. allColors fields

//\* limit

db.products.aggregate([

  { $match: { price: { $gt: 1200 } } },

  { $unwind: "$colors" },

  {

    $group: {

      \_id: { priceGroup: "$price" },

      allColors: { $addToSet: "$colors" },

    },

  },

  {

    $project: {

      \_id: 1,

      allColors: 1,

      colorLength: { $size: "$allColors" },

    },

  },

  { $limit: 1 },

]);

//\* skip

db.products.aggregate([

  { $match: { price: { $gt: 1200 } } },

  { $unwind: "$colors" },

  {

    $group: {

      \_id: { priceGroup: "$price" },

      allColors: { $addToSet: "$colors" },

    },

  },

  {

    $project: {

      \_id: 1,

      allColors: 1,

      colorLength: { $size: "$allColors" },

    },

  },

  { $skip: 1 },

]);

//\* $filter

db.col.aggregate([

  {

    $project: {

      name: 1,

      values: {

        $filter: {

          input: "$values",

          as: "value",

          cond: { $gt: ["$$value", 30] },

        },

      },

    },

  },

]);

Examples

db.produts.aggregate([

  {

    $group: {

      \_id: "$company",

      totalProducts: { $sum: "$price" },

    },

  },

]);

db.products.aggregate([

  {

    $match: {

      \_id: "64c23350e32f4a51b19b9247",

    },

  },

]);

price > 900

company $group

sum price

db.produts.aggregate([

    {

        $match: {price: {$gt: 900}}

    },

    {

      $group: {

        \_id: "$company",

        totalProducts: { $sum: "$price" },

      },

    },

  ]);

//!   find the quantity = 5, group them with same quantity and find the average price

db.sales.aggregate([

    { $match: {quantity:5} },

    {

        $group: {

            \_id: '$quantity',

            priceTotal: {$sum: '$price'},

            pricrAvg: {$avg:'$price'}

        }

    }

])

db.products.aggregate([

    { $match: { price: { $gt: 1200 } } },

    {

      $group: {

        \_id: "$category",

        totalPrice: { $sum: "$price" },

      },

    },

    {

        $sort: {totalPrice: 1}

    }

  ]);

  db.products.aggregate([

    { $match: { price: { $gt: 1200 } } },

    {

        $project: {

            price:1,

            discountPrice: {$multiply: ['$price', 0.8]}

        }

    }

  ])

  db.products.aggregate([

    { $match: { price: { $gt: 1200 } } },

    {

        $group: {

            \_id: '$price',

            allColors: { $push :'$colors'}

        }

    }

  ])

  price: 1999,

  colors: [ '#000000', '#cc6600', '#663300' ]

  price: 1999,

  colors: [ '#000000', '#cc6600', '#663300' ]

  price: 1999,

  colors: [

    [ '#000000', '#cc6600', '#663300' ],

    [ '#000000', '#cc6600', '#663300' ]

  ]

  ,

  price: 1999,

  colors: ['#000000', '#cc6600', '#663300']

  db.products.aggregate([

    { $unwind: '$colors' },

    { $match: { price: { $gt: 1200 } } },

    {

        $group: {

            \_id: '$price',

            allColors: { $push :'$colors'}

        }

    }

  ])

  db.products.aggregate([

    { $unwind: '$colors' },

    { $match: { price: { $gt: 1200 } } },

    {

        $group: {

            \_id: '$price',

            allColors: { $addToSet :'$colors'}

        }

    }

  ])

  db.products.aggregate([

    { $match: { price: { $gt: 1200 } } },

    { $unwind: "$colors" },

    {

      $group: {

        \_id: { priceGroup: "$price" },

        colors: { $addToSet: "$colors" },

      },

    },

    {

        $project:{

            \_id:1,

            colors:1,

            colorLength: { $size: "$colors" },

        }

    },

   {

    $limit: 1

   }

  ]);

  db.col.insertMany([

    {

      \_id: "64c23350e32f4a51b19b9201",

      name: "Document 1",

      values: [10, 20, 30, 40, 50],

    },

    {

      \_id: "64c23350e32f4a51b19b9202",

      name: "Document 2",

      values: [15, 25, 35, 45, 55],

    },

    {

      \_id: "64c23350e32f4a51b19b9203",

      name: "Document 3",

      values: [5, 15, 25, 35, 45],

    },

    {

      \_id: "64c23350e32f4a51b19b9204",

      name: "Document 4",

      values: [30, 40, 50, 60, 70],

    },

    {

      \_id: "64c23350e32f4a51b19b9205",

      name: "Document 5",

      values: [25, 35, 45, 55, 65],

    },

  ]

)

db.col.aggregate([{

    $project:{

        name:1,

        thapaValue: {

            $filter:{

                input: '$values',

                as: 'val',

                cond: {$gt: ['$$val', 30]}

            }

        }

    }

}])

Mongo Node

const { MongoClient } = require("mongodb");

const uri = "mongodb://127.0.0.1";

const client = new MongoClient(uri);

const data1 = {

  name: "Designer Handbag1",

  company: "64c23350e32f4a51b19b923a",

  price: 3466,

  colors: ["#000000", "#cc6600", "#663300"],

  image: "/images/product-handbag.png",

  category: "64c2342de32f4a51b19b9250",

  isFeatured: true,

};

const main = async () => {

  await client.connect();

  const db = client.db("shop");

  const collection = db.collection("products");

  await collection.insertOne(data1);

  const data = await collection.find({ price: { $eq: 3466 } }).toArray();

  console.log(data);

  return "done";

};

main()

  .then(console.log())

  .catch((e) => console.log(e))

  .finally(() => client.close());

Mongoose Node

const mongoose = require("mongoose");

// const uri = "mongodb://127.0.0.1/shop";

const uri =

  "mongodb+srv://vbthapa55:qwerty123@cluster0.kziyfmu.mongodb.net/shop?retryWrites=true&w=majority";

mongoose.connect(uri);

// we need to create a schema

const productSchema = new mongoose.Schema({

  name: String,

  company: String,

  price: Number,

  colors: [String],

  image: String,

  category: String,

  isFeatured: Boolean,

});

// we need to now create an model

const Product = new mongoose.model("Product", productSchema);

//? 2nd step while inserting the data

const data1 = {

  name: "Designer Handbag2",

  company: "64c23350e32f4a51b19b923a",

  price: 2466,

  colors: ["#000000", "#cc6600", "#663300"],

  image: "/images/product-handbag.png",

  category: "64c2342de32f4a51b19b9250",

  isFeatured: true,

};

const main = async () => {

  try {

    //? 2: insert documents

    await Product.insertMany(data1);

    const data = await Product.find({ price: { $eq: 2466 } });

    console.log(data);

    //? 3 update query

    await Product.findOneAndUpdate(

      { name: "Designer Handbag2", price: 2466 },

      { $set: { price: 2499 } }

    );

    //? 3 Delete query

    await Product.findOneAndDelete({ name: "Designer Handbag2", price: 2499 });

    const data = await Product.find({

      name: "Designer Handbag2",

      price: 2499,

    });

    console.log(data);

  } catch (error) {

    console.log(error);

  } finally {

    mongoose.connection.close();

  }

};

main();

db.Students.insertMany(

  [{

    name: "Binamra",

    age: 20,

  },

  {

    name: "Thapa",

    age: 21,

  }]

);

 {

    "name": "Binamra",

    "age": 20,

  }

  {

    "name": "Thapa",

    "age": 21,

  }

Element Operator

//\* Elements Operator

// In MongoDB, element operators are used to query documents based on the existence, type, and values of fields within the documents. These operators help you work with fields that are arrays, null, missing, or have specific data types.

//? 1: $exists: Matches documents that have a specific field, regardless of its value.

db.products.find({ price: { $exists: true } }).count();

// Find documents with the "price" field present, and if it's present then check the value greater then 1200

db.products.find({ price: { $exists: true },cls price: { $gt: 1250 } });

//? 2: $type: The $type operator filters documents based on the BSON data type of a field.

// Basically we need to search or find the fields based on types (BSON Type) for example

db.products.find({ price: { $type: "string" } });

// result will be 0, bcz the price type is number

db.products.find({ price: { $type: "number" } }).count()

// 1: Double

// 2: String

// 3: Object

// 4: Array

// 5: Binary data

// 6: Undefined

// 7: Object id

// 8: Boolean

// 9: Date

// 10: Null

// 11: Regular expression

// 12: JavaScript code

// 13: Symbol

// 14: JavaScript code with scope

// 17: 64-bit integer

db.products.find({ price: { $type: "string" } });

// result will be 0, bcz the price type is number

db.products.find({ price: { $type: "number" } }).count()

//? 3: $size: The $size operator matches documents where the size of an array field matches a specified value.

db.comments.find({comments: {$size:2}})

Mongo Import

? mongoimport E:\\mongo\products.json -d shop -c products

? mongoimport E:\\mongo\products.json -d shop -c products --jsonArray