.\3\25

LaB-2

1 ~~Perform the~~ .

1. create a collection by name Customers with the following attributes Cust_id , Acc_Bal, Acc_Type

use dbb
db.createCollection("Customers");

Insert at least 5 values into the table

2.
```
db.Customers.insertMany([
        { Cust_id : 1, Acc_Bal: 1500, Acc_Type: "z"},
        { Cust_id : 2, Acc_Bal: 800, Acc_Type: "y"},
        {Cust_id : 3, Acc_Bal : 2000, Acc_Type : "z"},
        {Cust_id : 4, Acc_Bal : 500, Acc_Type: "x"},
        {Cust_id : 5, Acc_Bal : 1800, Acc_Type: "z"}
   ]);
```

3-> Write a query to display those records whose total account balance is greater than 1200 of account type 'Z' for each customer_id.

```
=> db.Customers.find({ Acc_Bal: { $gt: 1200}, Acc_Type:
   "z" });
```

4 Determine Minimum and Maximum account balance for each Customer_id

```
db.Customers.aggregate([
      {
         $group:{
            _id :"$Cust_id",
            Min_Balance:{ $min : "$Acc_Bal"},
            Max_Balance: { $max: "$Acc_Bal" }
      }
 ]);   }
```

2. Yous are developing an e-commerce platform where users can browse and purchase products. Each product has a unique identifier, a name, a category, a price, and available quantity. Additionally, users can add products to their cart and place orders. Design a MongoDB schema to efficiently handle Product info, user carts, and orders.

Queries

→ Retrieve All Products :

```
> db.createCollection("Products");
> db.Products.insertMany([
  {product_id : "12345", name:"SmartPhone", category:
   "Electronics", Price:299.99, quantity:50},
     . . . . .
       . . . .
   ]);
> ~~db. Users.~~ (
> db.createCollection("users");
> db.Users.insertMany([
  { user_id : "789ghi",
      name :"John Doe",
      email : "John.doe@email.com",
      cart : [
          {product_id:"12345", quantity:2},
          {product_id:"23456", quantity:1}
        ],
        orders : [
            ],
   ]);
```

PAGE :
DATE :

```
>db.createCollection("orders");
>db.Orders.insertMany([
    {
        order_id:"order123",
        user_id:"789ghi",
        ],
    }
]);
```

1) Retrieve All Products:

```
db.Products.find({});
```

2) Retrieve Products in a Specific Category (e.g Electro-nics)

```
db.Products.find({category:"Electronics"});
```

3) Retrieve Products with Quantity Greater Than 0

```
db.Products.find({quantity:{$gt:0}});
```

4) Retrieve Products Sorted by Price in Ascending Order

```
db.Products.find({}).sort({price:1});
```

5) Retrieve Products with Price Less Than or equal to $100.

```
db.Product.find({price:100}});
```

Retrieve ~~Products~~ orders Placed by a User (user with ID "123abc")

7) Retrieve Products Added to a User's cart (User with ID "789ghi...")

```
db.Users.find({user_id:"123abc"}, {orders:1})
```

8. Retrieve Total Price of Orders placed by a User (user with ID "123abc...")

```
db.Users.aggregate([
  {$match: {user_id: "123abc..."}},
  {unwind: "$orders"},
  {$group: {
        _id : "$user_id",
        total price : {$sum: "$orders.total_price"}
   }
  }
]);
```

Additional Aggregation queries

1. Calculate Total Number of Products in Each Category.

```
db.Products.aggregate([
    {$group: {_id: "$category", total_products:
              {$sum: 1}}}
  ]);
```

2 Calculate Total Price of Products in each category.

```
db.Products.aggregate([
  { $group: {_id: "$category", total_price:
    {$sum: {$multiply: ["$price", "$quantity"]}}}
]);
```

PAGE :
DATE : / /

3 Find Average Price of Products

```
db.Products.aggregate([
    { $group: {_id: null, average_price:
        { $avg: "$price" }}}
```

4 Find Products with Quantity less than 10

```
db.Products.find({quantity: { $lt: 10}});
```

5 Sort Products by Price in Descending Order

```
db.Products.find({}).sort({price: -1});
```

6 Calculate Total Price of Order Placed by Each user

```
db.Orders.aggregate([
  { $group: {_id: "$user_id", total_spent:
  { $sum: "$total_price" }}}
  ]);
```

8 Find Average Total Price Orders

```
db.Orders.aggregate([
{ $group: {_id: null, average_order_price:
 {$avg: "$total_price" }}}
}];
```

[illegible]
11/3/25