LAB – 1

1.Perform the following DB operations using MongoDB.

1. Create a collection by name Customers with the following attributes.

Cust_id, Acc_Bal, Acc_Type

2. Insert at least 5 values into the table

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

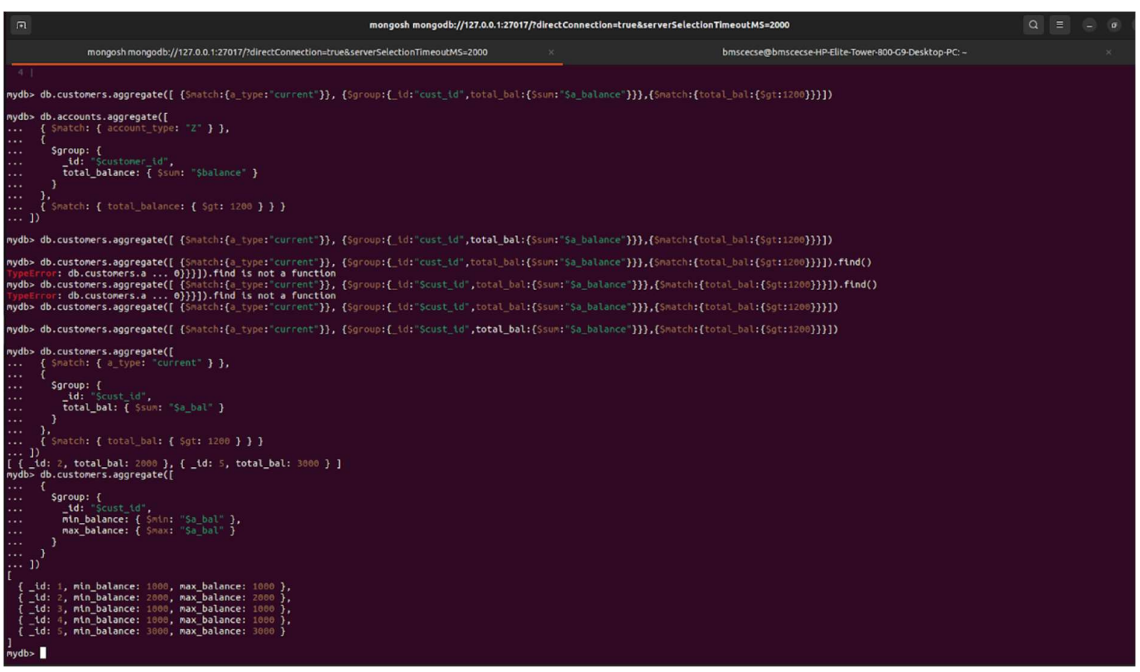
4. Determine Minimum and Maximum account balance for each customer_i

use mydb;

db.Customers.insertMany([{ Cust_id: "C001", Acc_Bal: 1000, Acc_Type: "A" }, { Cust_id: "C001", Acc_Bal: 500, Acc_Type: "Z" }, { Cust_id: "C001", Acc_Bal: 800, Acc_Type: "Z" }, { Cust_id: "C002", Acc_Bal: 1500, Acc_Type: "B" }, { Cust_id: "C002", Acc_Bal: 700, Acc_Type: "Z" }]);

db.Customers.aggregate([{ $match: { Acc_Type: "Z" } }, { $group: { _id: "$Cust_id",totalBalanceZ: { $sum: "$Acc_Bal" } } }, { $match: { totalBalanceZ: { $gt: 1200 }} }]);

db.Customers.aggregate([{ $group: { _id: "$Cust_id", // Group by Customer ID minAccBal: { $min: "$Acc_Bal" }, maxAccBal: { $max: "$Acc_Bal" } } }]);



2. You 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 information, user carts, and orders.

Query to

Retrieve All Products.

Retrieve Products in a Specific Category (e.g., Electronics).

Retrieve Products with Quantity Greater Than 0.

Retrieve Products Sorted by Price in Ascending Order.

Retrieve Products with Price Less Than or Equal to $100.

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

Retrieve Orders Placed by a User (User with ID "123abc...")

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

Additional Aggregation queries based on Assignment-3 design:

1 .Calculate Total Number of Products in Each Category.

2. Calculate Total Price of Products in Each Category.

3. Find Average Price of Products.

4. Find Products with Quantity Less Than 10.

5. Sort Products by Price in Descending Order.

6. Calculate Total Price of Orders Placed by Each User.

7. Find Users with the Highest Total Price of Orders.

8. Find Average Total Price of Orders.

mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000

mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000 × bmscecse@bmscecse-HP-Elite

```
 4 |

ydb> db.users.insert({
..   _id: ObjectId("60d5ec49f8d3e64f1c8b4567"),
..   name: "John Doe",
..   cart: [
..     {
..       product_id: ObjectId("60d5ec49f8d3e64f1c8b1234"),
..       quantity: 2
..     }
..   ]
.. })

 acknowledged: true,
 insertedIds: { '0': ObjectId('60d5ec49f8d3e64f1c8b4567') }

ydb> db.orders.insert({
..   user_id: ObjectId("60d5ec49f8d3e64f1c8b4567"), // example user_id
..   items: [
..     {
..       product_id: ObjectId("60d5ec49f8d3e64f1c8b1234"), // example product_id
..       quantity: 2,
..       price: 499.99
..     }
..   ],
..   total_price: 999.98,
..   order_date: ISODate("2025-05-23T00:00:00Z")
.. })

 acknowledged: true,
 insertedIds: { '0': ObjectId('68305071dfde9dd5b5c4c7b6') }

ydb> db.products.find()

 {
   _id: Code('function(t){return(0,c.assertArgsDefinedType)([t],[[void 0,"string","number","object"]],"ObjectId"),new e.ObjectId(t)}'),
   name: 'Laptop',
   category: 'Electronics',
   price: 999.99,
   quantity: 10
 }

ydb> db.products.find()

 {
   _id: Code('function(t){return(0,c.assertArgsDefinedType)([t],[[void 0,"string","number","object"]],"ObjectId"),new e.ObjectId(t)}'),
   name: 'Laptop',
   category: 'Electronics',
   price: 999.99,
   quantity: 10
 }

ydb> db.products.find({category:"Electronics"})
```

mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000

mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000 × bmscecse@bmscecse-HP-Elite-Tower-800-G9-Desktop-PC:

```
 {
   _id: Code('function(t){return(0,c.assertArgsDefinedType)([t],[[void 0,"string","number","object"]],"ObjectId"),new e.ObjectId(t)}'),
   name: 'Laptop',
   category: 'Electronics',
   price: 999.99,
   quantity: 10
 }

ydb> db.products.find({quantity:{$gt:0}})

 {
   _id: Code('function(t){return(0,c.assertArgsDefinedType)([t],[[void 0,"string","number","object"]],"ObjectId"),new e.ObjectId(t)}'),
   name: 'Laptop',
   category: 'Electronics',
   price: 999.99,
   quantity: 10
 }

ydb> db.products.find().sort({price:1})

 {
   _id: Code('function(t){return(0,c.assertArgsDefinedType)([t],[[void 0,"string","number","object"]],"ObjectId"),new e.ObjectId(t)}'),
   name: 'Laptop',
   category: 'Electronics',
   price: 999.99,
   quantity: 10
 }

ydb> db.products.aggregate([
..   { $group: { _id: "$category", total: { $sum: 1 } } }
.. ])
 { _id: 'Electronics', total: 1 } ]
ydb> db.products.aggregate([
..   { $group: { _id: "$category", total_price: { $sum: "$price" } } }
.. ])
 { _id: 'Electronics', total_price: 999.99 } ]
ydb> db.products.aggregate([
..   { $group: { _id: null, avg_price: { $avg: "$price" } } }
.. ])
 { _id: null, avg_price: 999.99 } ]
ydb> db.products.find({ quantity: { $lt: 10 } })

ydb> db.orders.aggregate([
..   { $group: { _id: "$user_id", total_spent: { $sum: "$total_price" } } },
..   { $sort: { total_spent: -1 } },
..   { $limit: 1 }
.. ])
 { _id: ObjectId('60d5ec49f8d3e64f1c8b4567'), total_spent: 999.98 } ]
ydb> db.orders.aggregate([
..   { $group: { _id: null, avg_order_price: { $avg: "$total_price" } } }
.. ])
 { _id: null, avg_order_price: 999.98 } ]
ydb>
```

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000                bmscecse@bmscecse-HP-
  { _id: 1, min_balance: 1000, max_balance: 1000 },
  { _id: 2, min_balance: 2000, max_balance: 2000 },
  { _id: 3, min_balance: 1000, max_balance: 1000 },
  { _id: 4, min_balance: 1000, max_balance: 1000 },
  { _id: 5, min_balance: 3000, max_balance: 3000 }
]
mydb> db.createCollection("shop")
{ ok: 1 }
mydb> db.createCollection("products")
{ ok: 1 }
mydb> db.createCollection("users")
{ ok: 1 }
mydb> db.createCollection("orders")
{ ok: 1 }
mydb> db.products.insert({
...   _id: ObjectId,
...   name: "Laptop",
...   category: "Electronics",
...   price: 999.99,
...   quantity: 10
... }
... )
{
  acknowledged: true,
  insertedIds: { '0': [Function (anonymous)] { help: [Function (anonymous)] Help } }
}
mydb> db.users.insert({
...   _id: ObjectId("789ghi..."),
...   name: "John Doe",
...   cart: [
...     {
...       product_id: ObjectId("..."),
...       quantity: 2
...     }
...   ]
... }
... )
BSONError: input must be a 24 character hex string, 12 byte Uint8Array, or an integer
mydb> db.users.insert({ _id: ObjectId("789"), name: "John Doe", cart: [ { product_id: ObjectId("..."), quantity: 2 }] } )
mydb> db.users.insert({ _id: ObjectId(""), name: "John Doe", cart: [ { product_id: ObjectId(""), quantity: 2 }] } )
BSONError: input must be a 24 character hex string, 12 byte Uint8Array, or an integer
mydb> db.users.insert({ _id: ObjectId(""), name: "John Doe", cart: [ { product_id: ObjectId(""), quantity: 2 }] }
...
... db.users.insert({ _id: ObjectId(""), name: "John Doe", cart: [ { product_id: ObjectId(""), quantity: 2 }] }
Uncaught:
SyntaxError: Unexpected token, expected "," (3:0)

  1 | db.users.insert({ _id: ObjectId(""), name: "John Doe", cart: [ { product_id: ObjectId(""), quantity: 2 }] }
  2 |
> 3 | db.users.insert({ _id: ObjectId(""), name: "John Doe", cart: [ { product_id: ObjectId(""), quantity: 2 }] }
    | ^
  4 |
```

```
mydb> db.customers.aggregate([ {$match:{a_type:"current"}}, {$group:{_id:"cust_id",total_bal:{$sum:"$a_balance"}}},{$match:{total_bal:{$gt:1200}}}])

mydb> db.accounts.aggregate([
...   { $match: { account_type: "Z" } },
...   {
...     $group: {
...       _id: "$customer_id",
...       total_balance: { $sum: "$balance" }
...     }
...   },
...   { $match: { total_balance: { $gt: 1200 } } }
... ])

mydb> db.customers.aggregate([ {$match:{a_type:"current"}}, {$group:{_id:"cust_id",total_bal:{$sum:"$a_balance"}}},{$match:{total_bal:{$gt:1200}}}])

mydb> db.customers.aggregate([ {$match:{a_type:"current"}}, {$group:{_id:"cust_id",total_bal:{$sum:"$a_balance"}}},{$match:{total_bal:{$gt:1200}}}]).find()
TypeError: db.customers.a ... 0}}}]).find is not a function
mydb> db.customers.aggregate([ {$match:{a_type:"current"}}, {$group:{_id:"$cust_id",total_bal:{$sum:"$a_balance"}}},{$match:{total_bal:{$gt:1200}}}]).find()
TypeError: db.customers.a ... 0}}}]).find is not a function
mydb> db.customers.aggregate([ {$match:{a_type:"current"}}, {$group:{_id:"$cust_id",total_bal:{$sum:"$a_balance"}}},{$match:{total_bal:{$gt:1200}}}])

mydb> db.customers.aggregate([ {$match:{a_type:"current"}}, {$group:{_id:"$cust_id",total_bal:{$sum:"$a_balance"}}},{$match:{total_bal:{$gt:1200}}}])

mydb> db.customers.aggregate([
...   { $match: { a_type: "current" } },
...   {
...     $group: {
...       _id: "$cust_id",
...       total_bal: { $sum: "$a_bal" }
...     }
...   },
...   { $match: { total_bal: { $gt: 1200 } } }
... ])
[ { _id: 2, total_bal: 2000 }, { _id: 5, total_bal: 3000 } ]
mydb> db.customers.aggregate([
...   {
...     $group: {
...       _id: "$cust_id",
...       min_balance: { $min: "$a_bal" },
...       max_balance: { $max: "$a_bal" }
...     }
...   }
... ])
[
  { _id: 1, min_balance: 1000, max_balance: 1000 },
  { _id: 2, min_balance: 2000, max_balance: 2000 },
  { _id: 3, min_balance: 1000, max_balance: 1000 },
  { _id: 4, min_balance: 1000, max_balance: 1000 },
  { _id: 5, min_balance: 3000, max_balance: 3000 }
]
```