GROUP PROJECT- PHASE2

For

George Wanganga Instructor

of *Database design and implementation*

Sheridan College

Brampton, Ontario

December 6,2023

Part 1

Use the order.json file to answer these two questions:

1. Calculate Total Order Quantity:

QUERY:

```
db.getCollection("movieDetails").find({ "year": { "$lte": 1970 } })
// Calculate Total Order Quantity
var totalOrderQuantity = db.orders.aggregate([
  {
    $group: {
      _id: null,
      totalQuantity: { $sum: "$quantity" }
    }
  }
]).toArray();
print("Total Order Quantity: ", totalOrderQuantity[0].totalQuantity);
```

SNAPSHOTS:

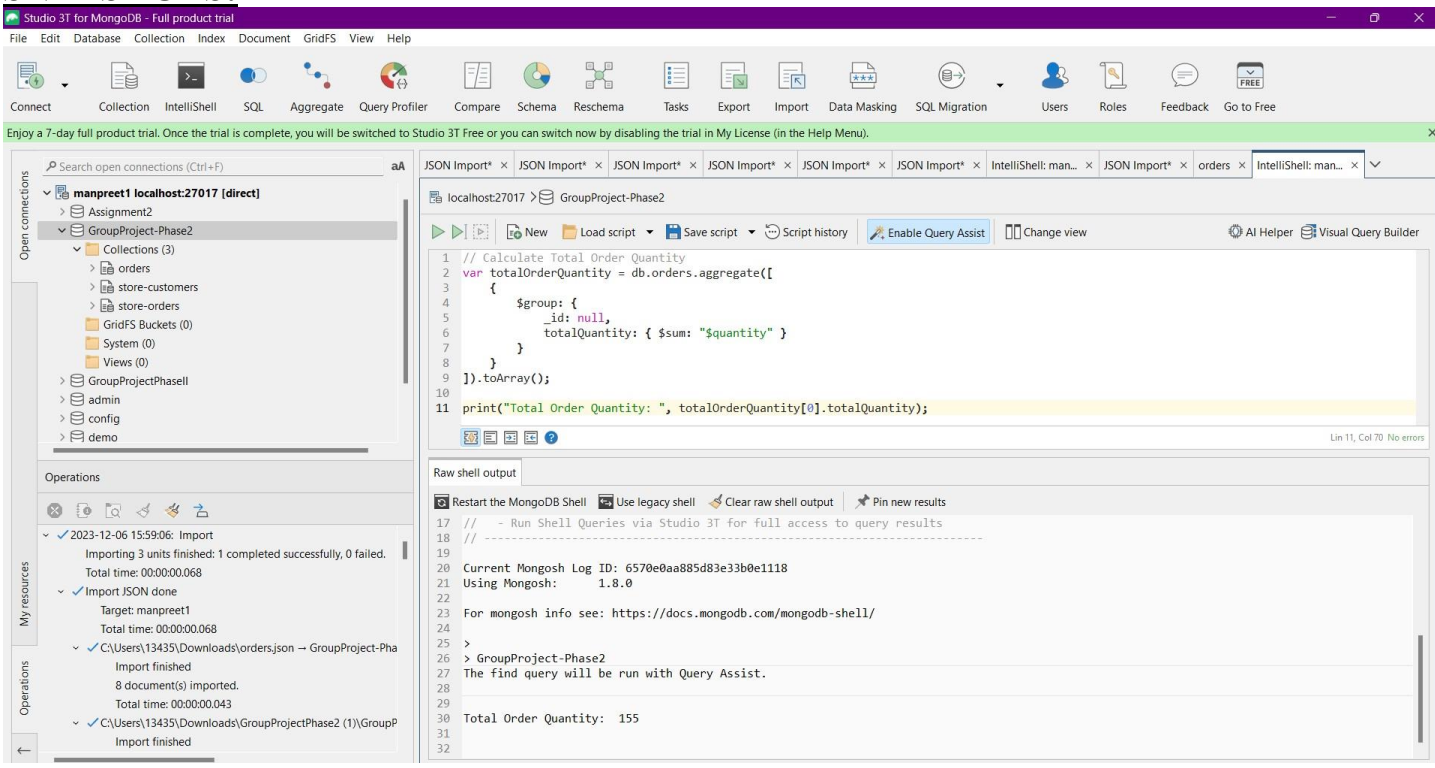

Figure 1: to show the successful execution of the query.

2. Calculate Total Order Value and Average Order Quantity

QUERY:

```
// Calculate Total Order Value and Average Order Quantity
var orderStats = db.orders.aggregate([
    {
        $group: {
            _id: null,
            totalValue: { $sum: { $multiply: ["$price", "$quantity"] } },
            averageQuantity: { $avg: "$quantity" }
        }
    }
]).toArray();

print("Total Order Value: ", orderStats[0].totalValue);
print("Average Order Quantity: ", orderStats[0].averageQuantity);
```

SNAPSHOTS:

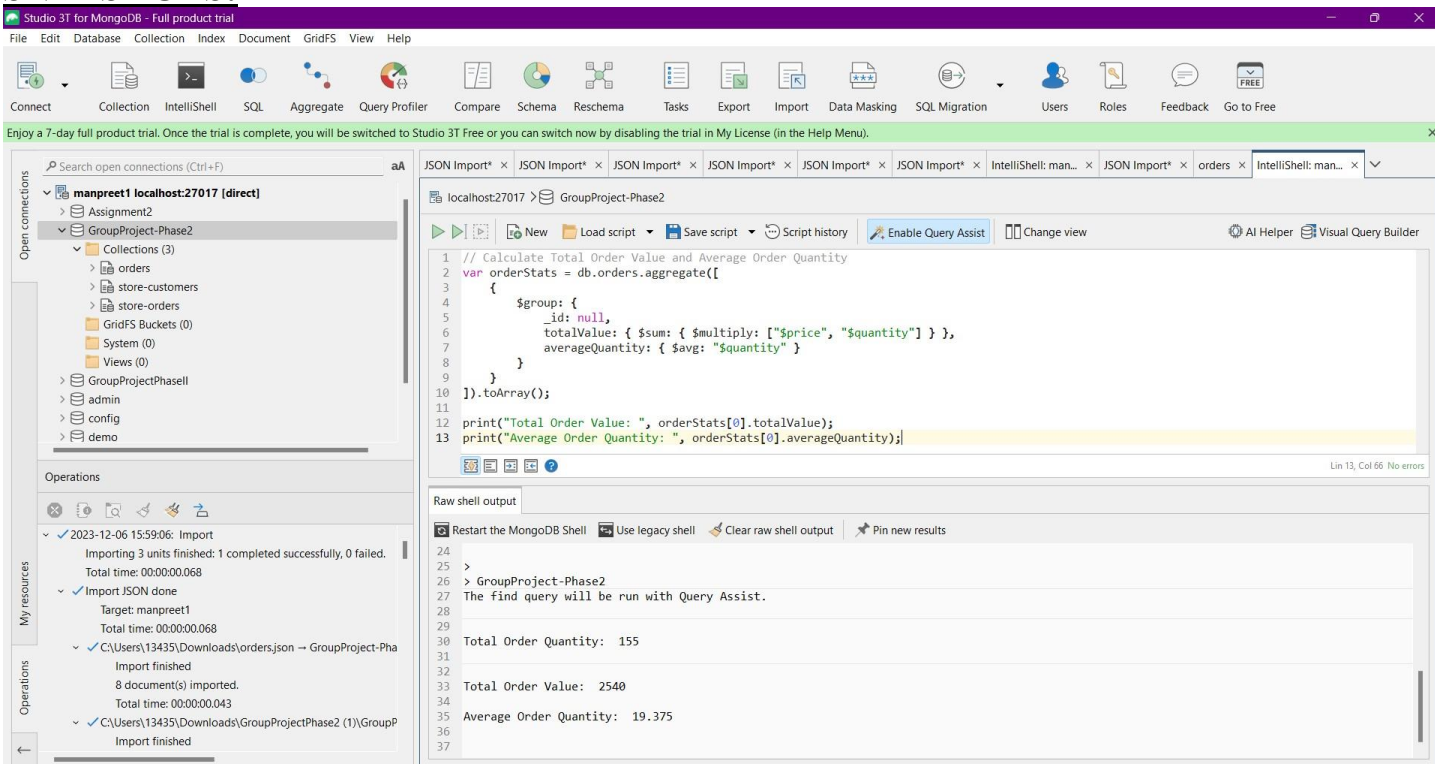

Figure 2 : to show the successful execution of the query.

Use the these files (store-customers.json and store-orders.json) to answer the following questions:

1. **Count of all active customers**

QUERY:

```
db.getCollection("store-customers").find({ active: true}).pretty();
```

SNAPSHOTS:

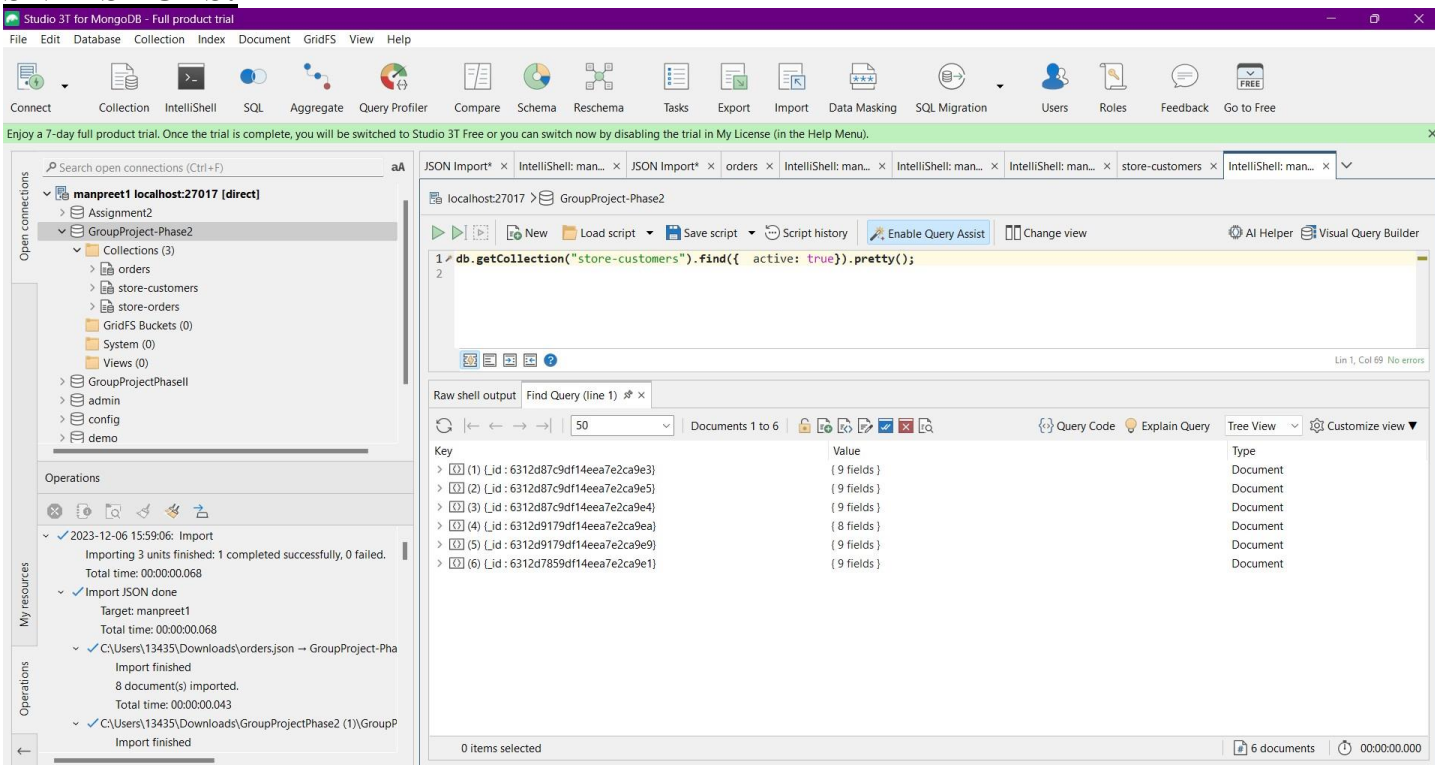

Figure 3 : to show the successful execution of the query.

2. Return a list of every date in the year 2022, with the total amount of all orders placed on that date, sorted by date (earliest first).

QUERY:

```
db.getCollection("store-orders").aggregate([

{

$match: {

"date": {

$gte: ISODate("2022-01-01T00:00:00Z"),

$lt: ISODate("2023-01-01T00:00:00Z")}}},

{

$group: {

_id: { $dateToString: { format: "%Y-%m-%d", date: "$date" } },

totalAmount: { $sum: { $toDouble: "$amount" } }}},

{

$sort: { _id: 1 }

}

]).forEach(function (result) {

print("Date: " + result._id + ", Total Amount: " + result.totalAmount);

});
```

SNAPSHOTS:

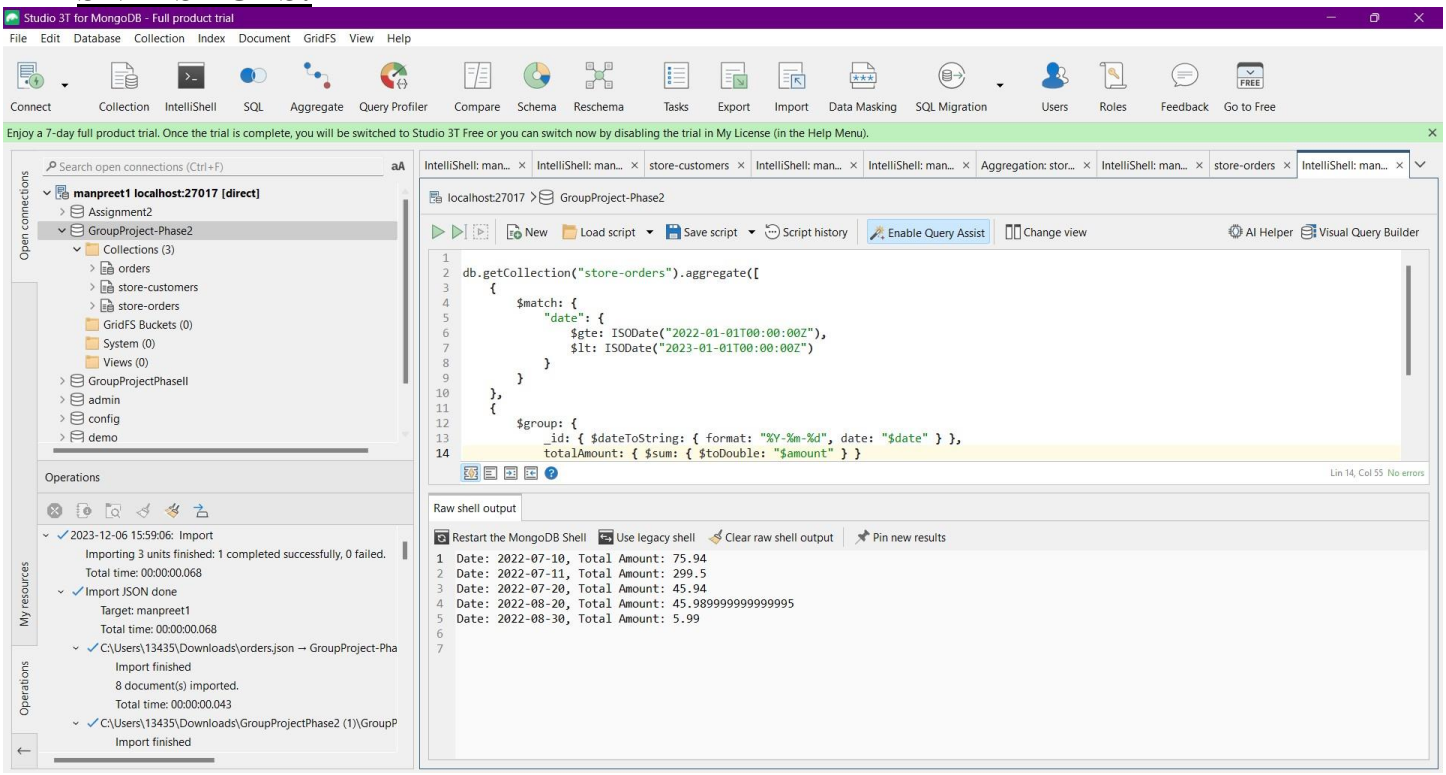

Figure 4: to show the successful execution of the query.

3. All customers, but shape the output by only returning their name and favorite categories.

- Same query ... but this time, rename the property `favoriteCategories` to `faves`

QUERY:

```
db.getCollection("store-customers").aggregate([
{
$project: {
_id: 0,
name: 1,
faves: "$favoriteCategories"
}
}
])
```

SNAPSHOTS:

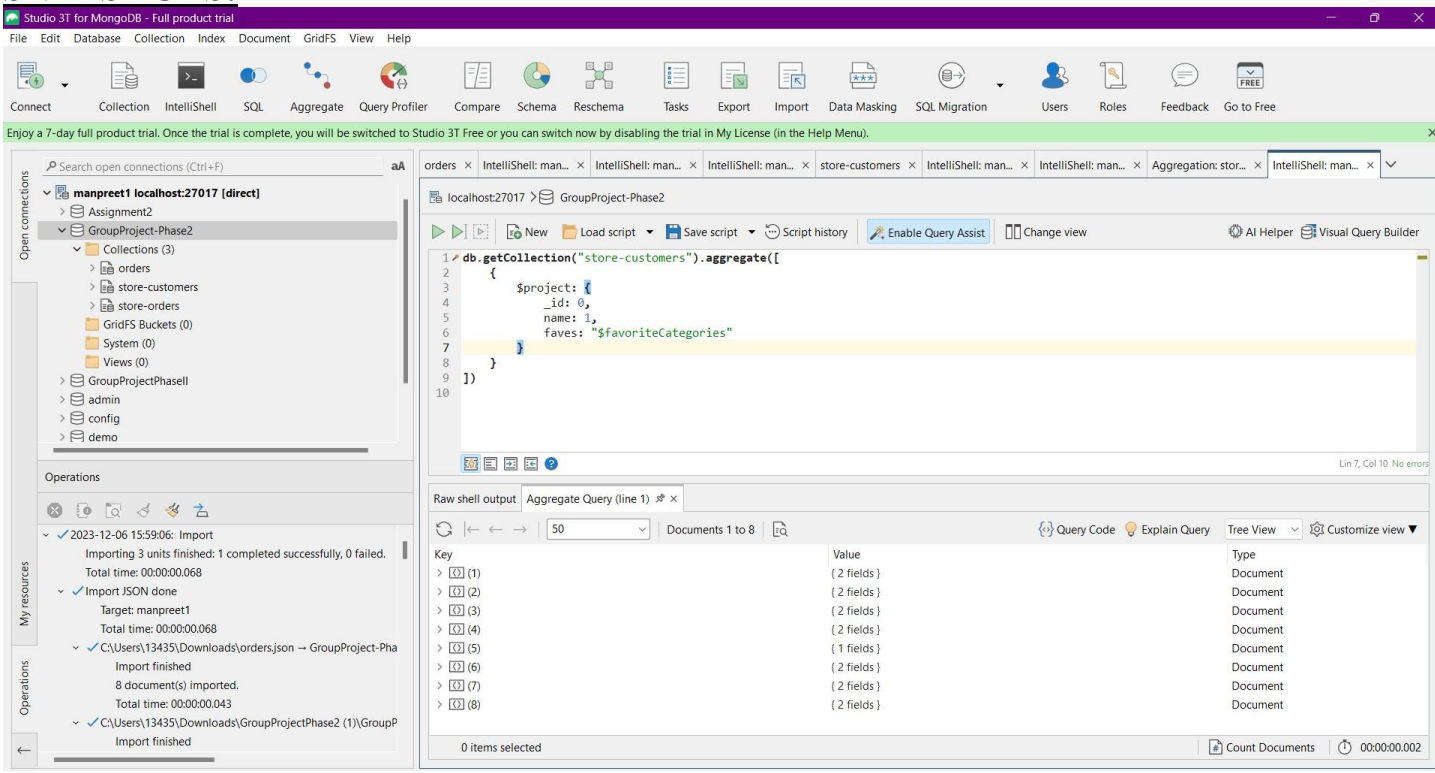

Figure 5: to show the successful execution of the query.

4. All customers, but shape the output by only returning their name and the number of `favoriteCategories` for each
- The average number of customers `favoriteCategories`, grouped by the `state` for their billing address.

QUERY:

```
db.getCollection("store-customers").aggregate([
{
$match: {
favoriteCategories: { $exists: true, $type: "array" },
"addresses.billing.state": { $exists: true }
}
},
{
$project: {
```

```
_id: 0,
name: 1,
numberOfFavorites: { $size: "$favoriteCategories" },
"addresses.billing.state": 1
}
},
{
$group: {
_id: "$addresses.billing.state",
avgFavorites: { $avg: "$numberOfFavorites" },
customers: { $push: { name: "$name", numberOfFavorites: "$numberOfFavorites" } }
}
}
])
```

SNAPSHOTS:

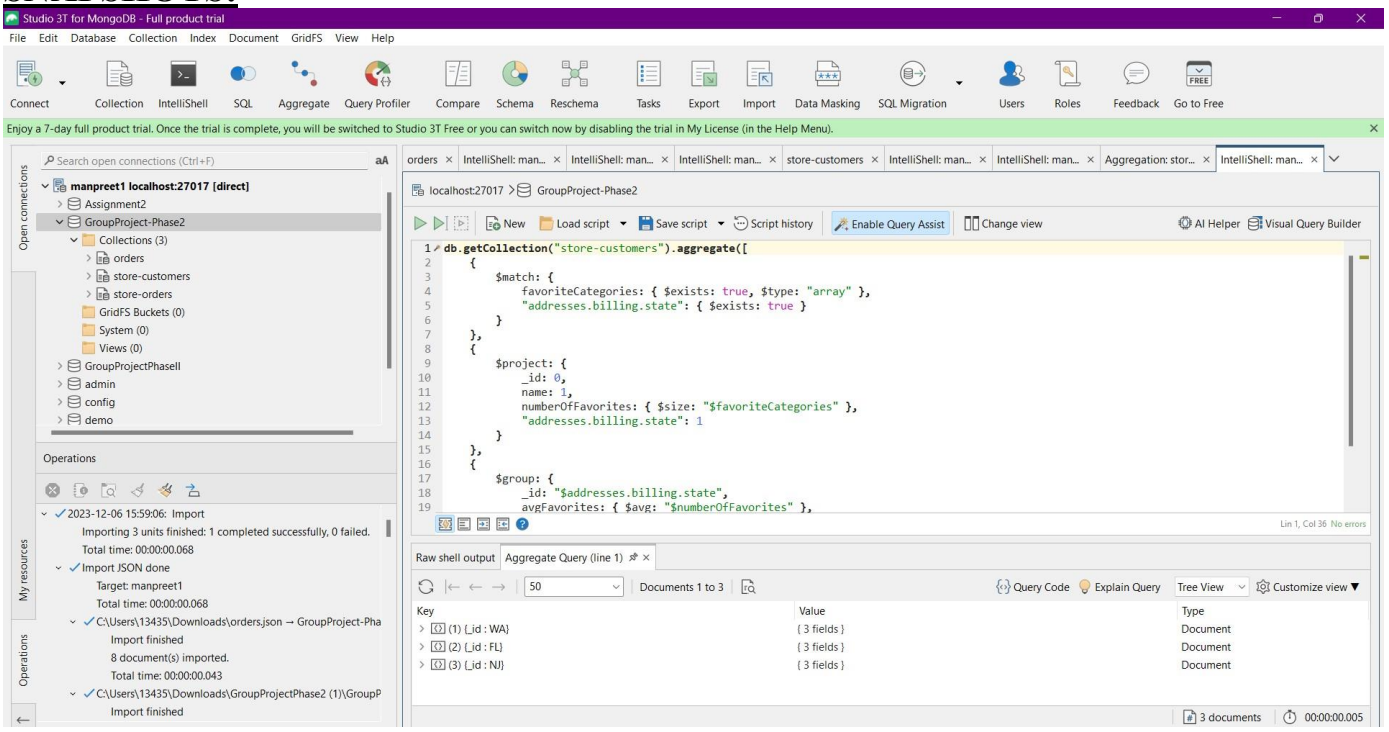

Figure 6: to show the successful execution of the query.

5. Using "normal" aggregation, return the average number of customer `favoriteCategories`, grouped by `state`

QUERY:

```
db.getCollection("store-customers").aggregate([
{
$match: {
favoriteCategories: { $exists: true, $type: "array" },
"addresses.billing.state": { $exists: true }
}
},
{
$project: {
_id: 0,
numberOfFavorites: { $size: "$favoriteCategories" },
"addresses.billing.state": 1
}
},
{
$group: {
_id: "$addresses.billing.state",
avgFavorites: { $avg: "$numberOfFavorites" }
}
}
])
```

SNAPSHOTS:

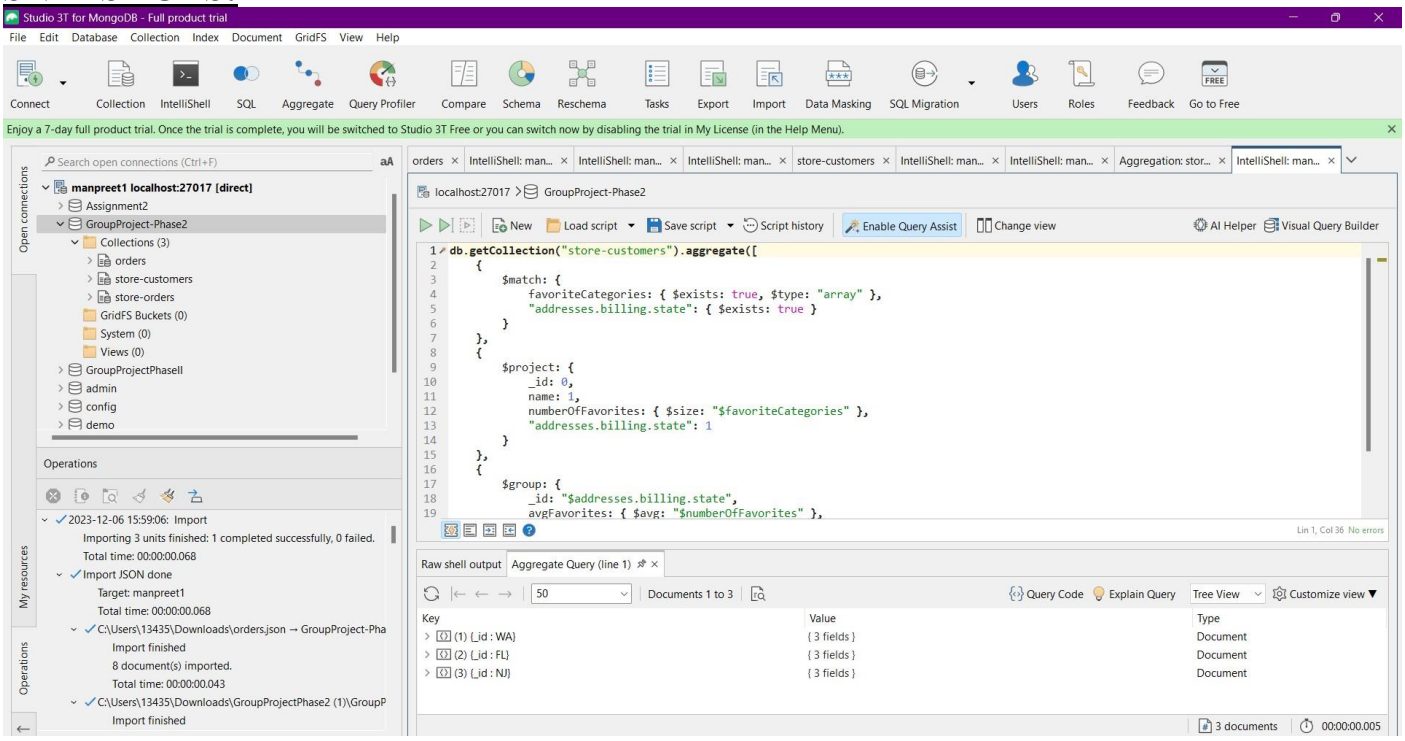

Figure 7: to show the successful execution of the query.

-Repeat that query, but use an $accumulator to perform the calculation

QUERY:

```
db.getCollection("store-customers").aggregate([
{
$match: {
favoriteCategories: { $exists: true, $type: "array" },
"addresses.billing.state": { $exists: true }
}
},
{
$project: {
_id: 0,
numberOfFavorites: { $size: "$favoriteCategories" },
"addresses.billing.state": 1
}
},
{
$group: {
_id: "$addresses.billing.state",
avgFavorites: {
$accumulator: {
init: function() {
return { count: 0, total: 0 };
},
accumulate: function(state, numFavorites) {
return {
count: state.count + 1,
total: state.total + numFavorites
};
},
accumulateArgs: ["$numberOfFavorites"],
merge: function(state1, state2) {
return {
count: state1.count + state2.count,
total: state1.total + state2.total
};
},
finalize: function(state) {
return state.total / state.count;
},
lang: "js"
}
}
}
}
])
])
```

SNAPSHOTS:

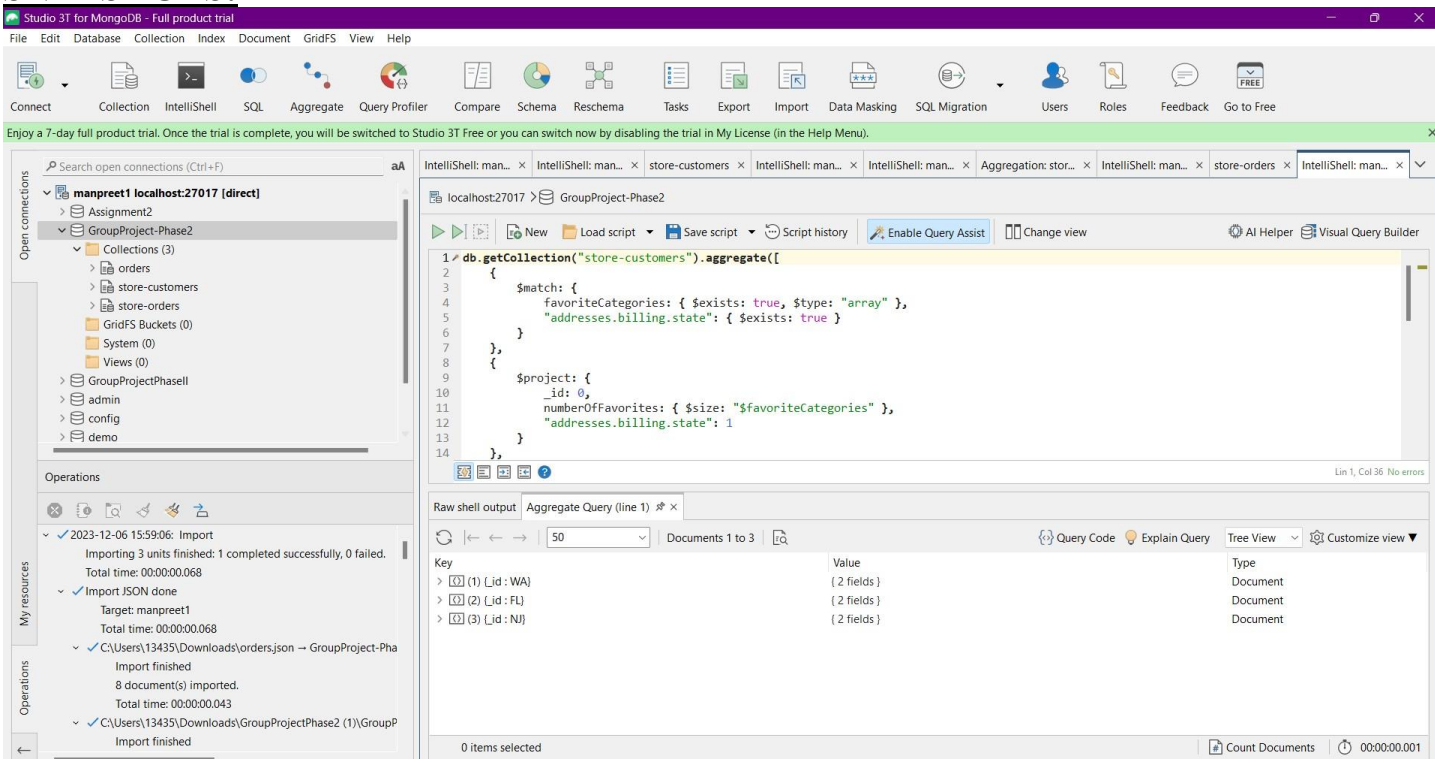

Figure 8: to show the successful execution of the query.

6. Return a list of customers and the the count of their `favoriteCategories` - The output should contain only the customer's name and a field called `numberOfFavorites

QUERY:

```
db.getCollection("store-customers").aggregate([
{
$project: {
_id: 0,
name: 1,
numberOfFavorites: { $size: {$ifNull: ["$favoriteCategories", []]}}
}}
]);
```

SNAPSHOTS:

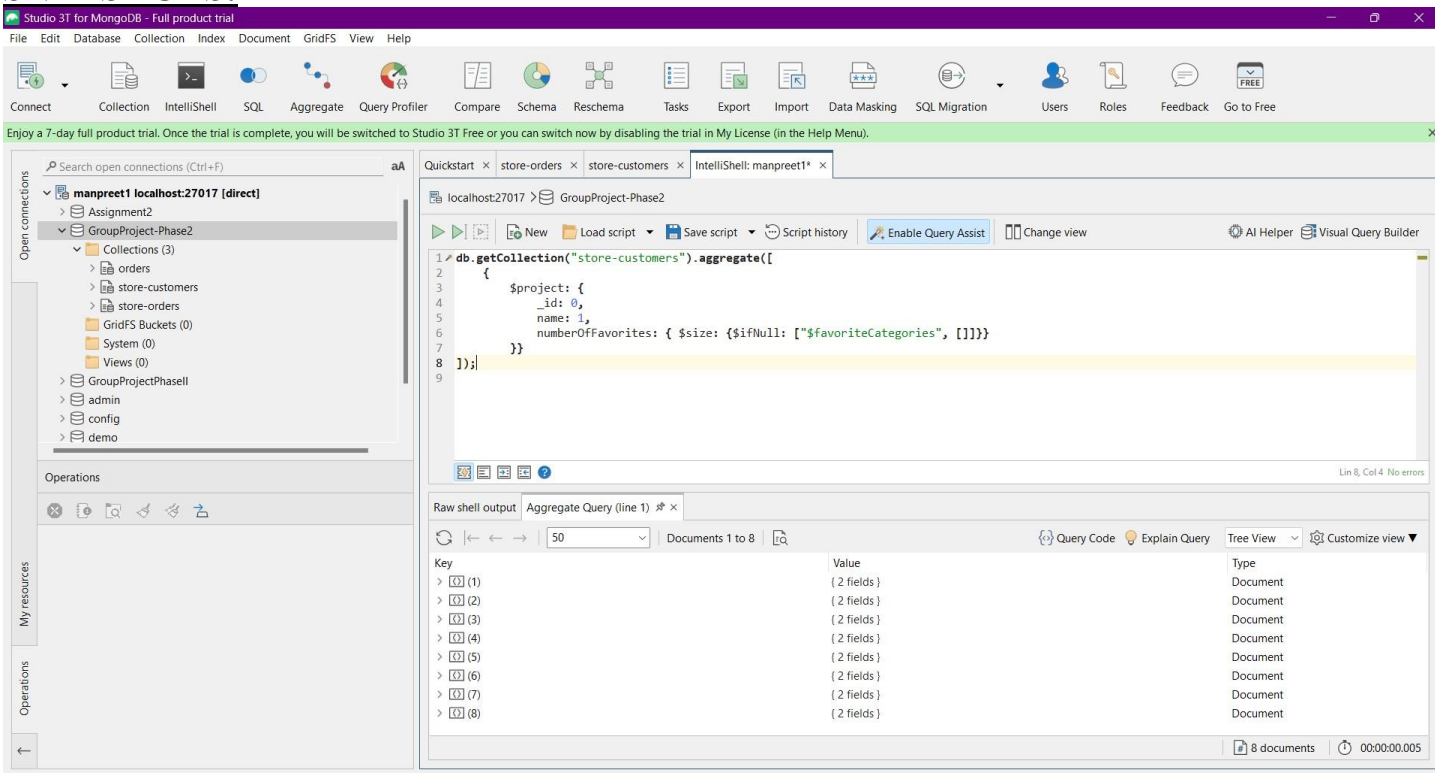

Figure 9: to show the successful execution of the query.

7. Return the full customer document associated with an order, given an order id - Use `findOne()` to obtain a customer id from an order and then use that to return a customer using `findOne()`) with that ID.

QUERY:

```
var orderId = "6312eec6f80e3117f621a463"; // Replace this with the actual order id
// Find the order document to obtain the customer id
var order = db.getCollection("store-orders").findOne({ "_id": ObjectId(orderId) });
if (order) {
// Find the customer document using the obtained customer id
var customerId = order.customer;
var customer = db.getCollection("store-customers").findOne({ "_id": ObjectId(customerId) });
if (customer) {
printjson(customer);
} else {
print("Customer not found");
}
} else {
print("Order not found");
}
```

SNAPSHOTS:

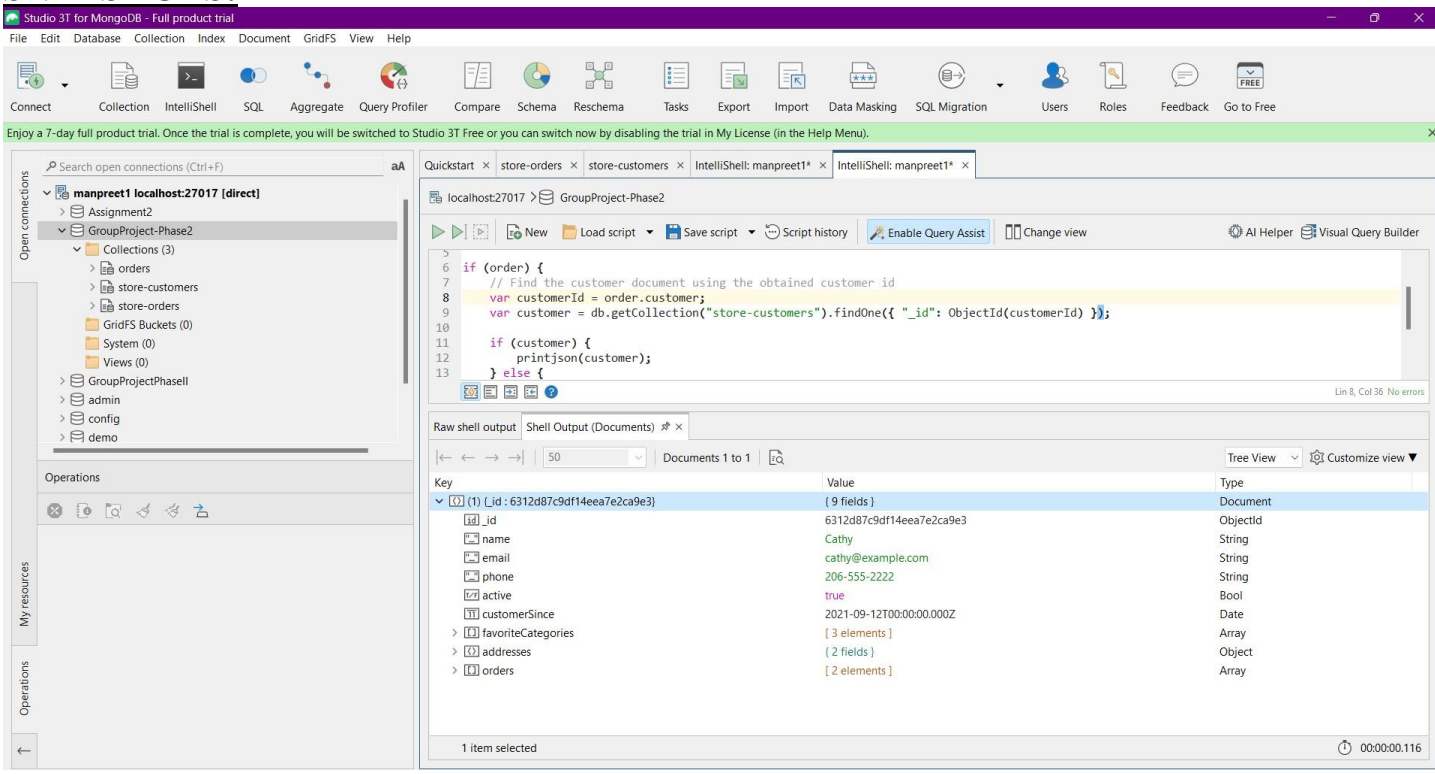

Figure 9: to show the successful execution of the query.

8. Return a list of all customers and their orders.
- The data should contain only the customer's name along with the list of their orders.
-Return a list of all orders and their associated customer information.
- Optional: use $project in the customer lookup to limit the amount of customer information returned

QUERY:

```
db.getCollection("store-customers").aggregate([
{
$lookup: {
from: "orders",
localField: "_id",
foreignField: "customer.$oid",
as: "orders"
}
},
{
$project: {
_id: 1,
name: 1,
orders: {
```

```
_id: 1,
amount: 1,
date: 1
// Add more fields as needed
}
}
}
]).forEach(printjson);
```

SNAPSHOTS:

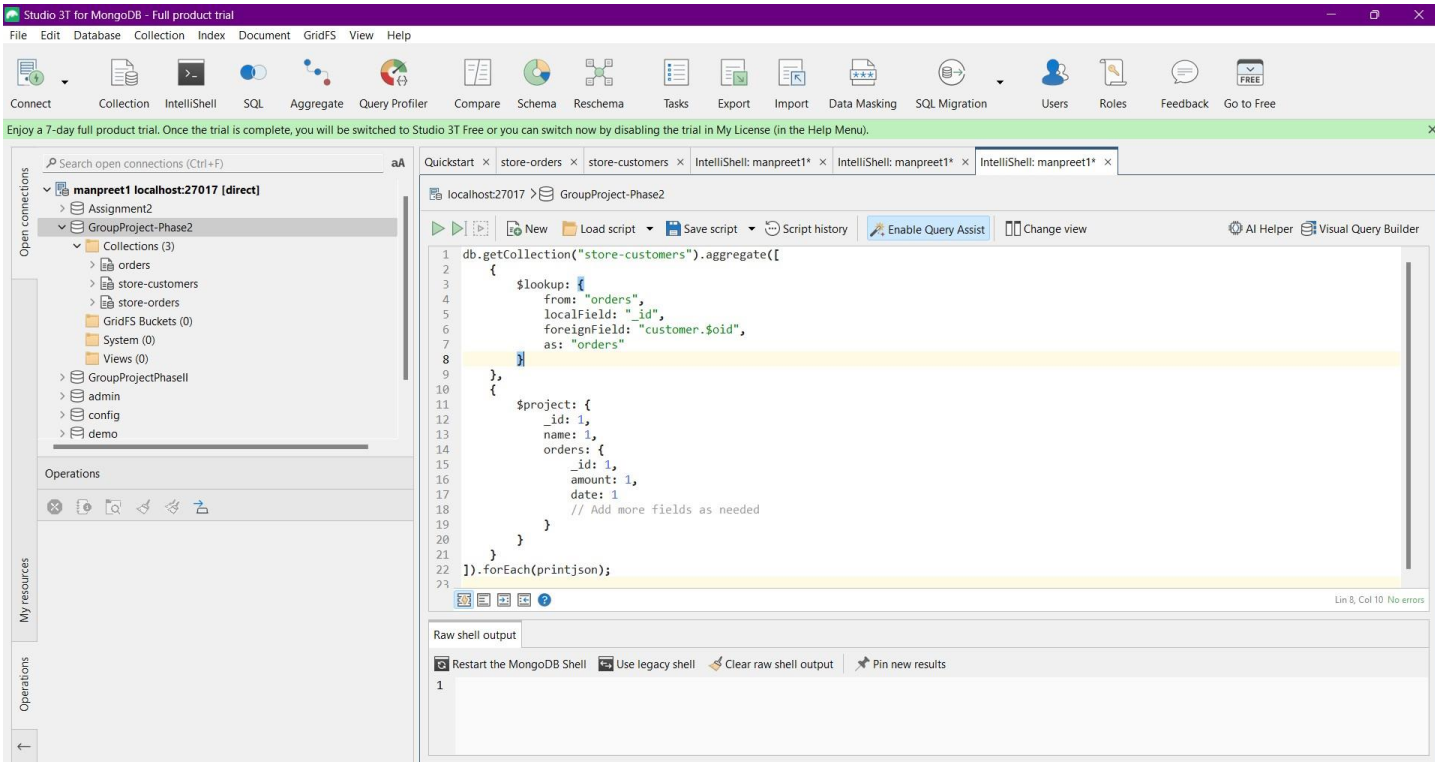

Figure 10: to show the successful execution of the query.

9. Return a list of all orders, joined with their associated customer.

- The list returned should be a full, unfiltered and unshaped list of all order and customer properties.

QUERY:

```
db.getCollection("store-customers").aggregate([
{
$lookup: {
from: "customers",
localField: "customer.$oid",
foreignField: "_id",
as: "customer"
}
},
```

```
{
$unwind: "$customer"
},
{
$replaceRoot: { newRoot: { $mergeObjects: ["$customer", "$$ROOT"] } }
},
{
$project: { customer: 0 }
}
]).forEach(printjson);
```

SNAPSHOTS:

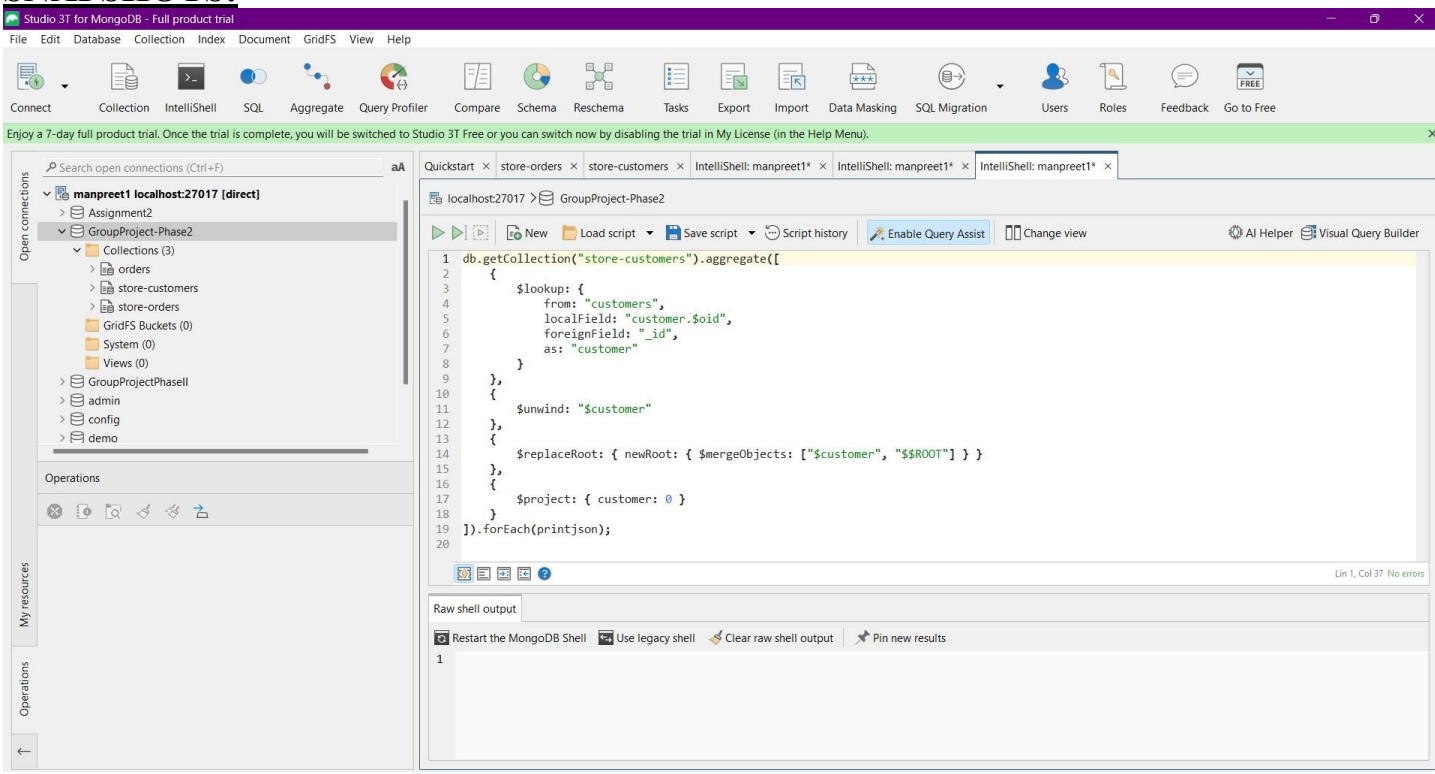

Figure 11: to show the successful execution of the query.