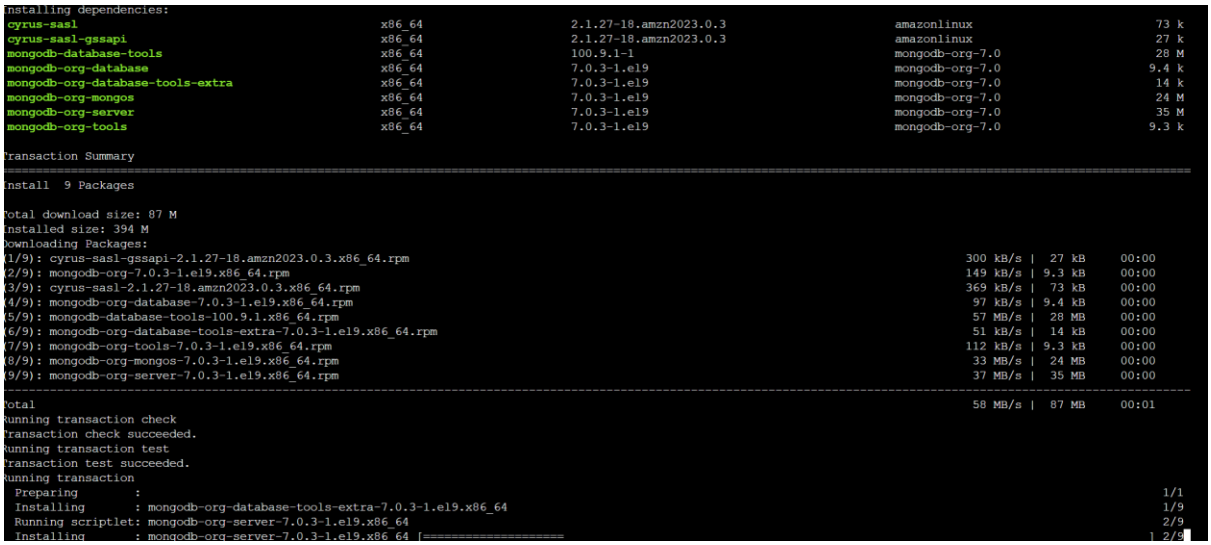
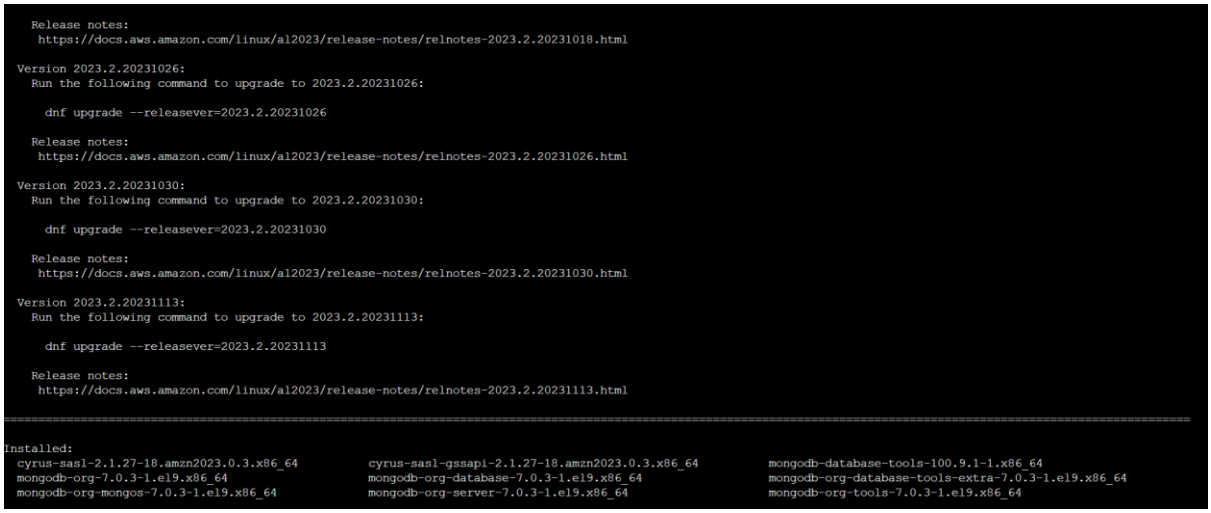
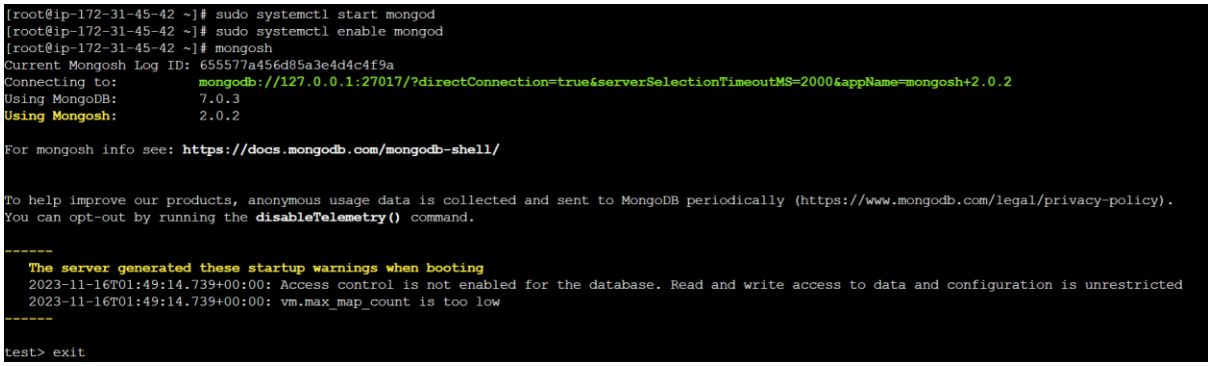
MongoDB Milestone

Q1. Installation of Mongodb-org





Mongosh:



(g) We replaced the $releaseversion with 9 to make sure that the latest version of our Operating system, Redhat Linux repository was only used for installation instead of MongoDB's default Linux repository. We did not get any values when we had just given $releaseversion as the repository was not compatible with Linux Version we were using.

Q2. Importing the cust1.json and placing the documents into collection "Customers"

```
[root@ip-172-31-45-42 ~]# mongoimport --db pos --collection Customers /var/lib/mysql/pos/cust1.json
2023-11-16T02:25:34.721+0000    connected to: mongodb://localhost/
2023-11-16T02:25:34.903+0000    7986 document(s) imported successfully. 0 document(s) failed to import.
```

Q3. Importing the cust2.json and placing the documents into collection "CustomerOrders"

```
[root@ip-172-31-45-42 ~]# mongoimport --db pos --collection CustomerOrders /var/lib/mysql/pos/cust2.json
2023-11-16T02:29:13.706+0000    connected to: mongodb://localhost/
2023-11-16T02:29:14.895+0000    7986 document(s) imported successfully. 0 document(s) failed to import.
```

Q4. Importing the ord.json and placing the documents into collection "Orders"

```
[root@ip-172-31-45-42 ~]# mongoimport --db pos --collection Orders /var/lib/mysql/pos/ord.json
2023-11-16T02:31:38.967+0000    connected to: mongodb://localhost/
2023-11-16T02:31:40.304+0000    24468 document(s) imported successfully. 0 document(s) failed to import.
```

Q5. Importing the prod.json and placing the documents into collection "Products"

```
[root@ip-172-31-45-42 ~]# mongoimport --db pos --collection Products /var/lib/mysql/pos/prod.json
2023-11-16T02:33:07.479+0000    connected to: mongodb://localhost/
2023-11-16T02:33:08.081+0000    4022 document(s) imported successfully. 0 document(s) failed to import.
```

(c) mongoimport is used to import data from the files (In this case, JSON Files) to the mongodb database. In the previous milestone, we wrote in JSON Format because it could be easily imported in mongodb in Collections. Also we could perform complex queries and aggregations through JSON which we used in this milestone.

Q6.

(a)

```
pos> db.Customers.find({ "Customer Address":  /TX/i  })
[
  {
    _id: ObjectId("65557d9ed640697d7c64b3c3"),
    'Customer Name': 'Alexander Perry',
    'Customer Address': '9128 Clover St\\n SABINAL TX 78881'
  },
  {
    _id: ObjectId("65557d9ed640697d7c64b3cd"),
    'Customer Name': 'Raj Manson',
    'Customer Address': '4635 Turkey Cir\\n BIVINS TX 75555'
  },
  {
    _id: ObjectId("65557d9ed640697d7c64b3e8"),
    'Customer Name': 'Valentina Kattimani',
    'Customer Address': '699 Parsley Road\\n CHESTER TX 75936'
  },
  {
    _id: ObjectId("65557d9ed640697d7c64b3e9"),
    'Customer Name': 'Sarah Gain',
    'Customer Address': '9179 Fig Road\\n ARP TX 75750'
  },
  {
    _id: ObjectId("65557d9ed640697d7c64b3fc"),
    'Customer Name': 'Luca Lamb',
    'Customer Address': '1577 Ibis Way\\n TYLER TX 75701'
  },
  {
    _id: ObjectId("65557d9ed640697d7c64b418"),
    'Customer Name': 'Parker Kumar',
    'Customer Address': '4361 Violet Ln\\n CUT AND SHOOT TX 77306'
  },
  {
    _id: ObjectId("65557d9ed640697d7c64b41e"),
    'Customer Name': 'Aj Kannoth',
    'Customer Address': '4414 Honey Locust SW\\n LEONARD TX 75452'
```

```
pos> db.Customers.count({ "Customer Address":  /TX/i  })
DeprecationWarning: Collection.count() is deprecated. Use countDocuments or estimatedDocumentCount.
492
```

(b) The “db.Customers.Find” query works like the Select query in SQL. In this query we use the find function to find the Customers who live in the state of Texas (TX). The “I” at the end is for case insensitive. This means that even if in the address someone has written their query as “tx”, it would recognise them as residents of Texas. I have chosen the Customers Collection, because we just want

the names of those Customers, and the Collection only consists of Customer Name and Address which is ideal for this situation.

The second query was written to verify the number of Customers who resided in TX. The number matched the number of Json Objects in the JSON document.

Q7 Best Customer:

```
pos> db.CustomerOrders.aggregate([
...   {
...     $unwind: "$Order Placed"
...   },
...   {
...     $group: {
...       _id: "$Customer Name",
...       maxOrderTotal: { $sum: "$Order Placed.OrderTotal" }
...     }
...   },
... {
...  $sort: {maxOrderTotal: -1}
... },
... {
...   $limit:1
... }
... ]);
[ { _id: 'Virginia Cooper', maxOrderTotal: 46392.93 } ]
```

(b) My criteria for the Best Customer was to Choose a customer whose OrderTotal is the highest. In the above query, I have firstly performed the aggregate function. In this function, I could perform various operations like $sum and $sort. I have first taken the sum of OrderTotal of all the orders placed by the customer. This gave me maxOrderTotal which was the collective sum of the price of all the orders placed by the customer. Then I sorted the maxOrderTotal in decreasing order (-1 stands for decreasing). The “$limit:1” I have then written to show me the top Customer in the List who has the maximum OrderTotal. I chose the “CustomerOrders” collection because I needed a collection which consisted of all the orders a Customer has placed. In SQL, I would have joined the Customer table with the Order Table.

Q8 Best Product:

```
pos> db.Products.aggregate([
...   {
...     $unwind: "$Customer"
...   },
...   {
...     $group: {
...       _id: "$Product ID",
...       maxcustomers: { $sum:1 }
...     }
...   },
... {
...  $sort: {maxcustomers: -1}
... },
... {
...   $limit:1
... }
... ]);
[ { _id: 2266, maxcustomers: 36 } ]
```

(b) My best Product would be the Product which have the maximum number of Customers. In this query , I have taken the Products collection, unwinded the Customer Array and counted all the Objects in the array (sum:1 is used to count). This way I counted all the Customers which bought their product. Then I sorted the maxcustomers in decreasing order and put a limit of 1 to display the top product who has the maximum number of Customers.The result displayed the Product-ID and the number of customers who have bought those products. I have used the Products collection to check all the Products and their customers.

Q9. Which Product Needs to be discarded

(a)

```
pos> db.Products.aggregate([
...   {
...     $unwind: "$Customer"
...   },
...   {
...     $group: {
...       _id: "$Product ID",
...       mincustomers: { $sum:1 }
...     }
...   },
... {
...  $sort: {mincustomers: 1}
... },
... {
...   $limit:5
... }
... ]);
[
  { _id: 1836, mincustomers: 6 },
  { _id: 700, mincustomers: 6 },
  { _id: 3026, mincustomers: 7 },
  { _id: 2553, mincustomers: 7 },
  { _id: 4013, mincustomers: 7 }
]
```

(b)

The Products which have been bought by the least number of Customers needs to be discarded. In the above query I have unwinded the Customer Array and counted all the Customers in the array(similar to the above query). Then I am sorting the mincustomers in ascending order(:1 is for ascending) and found the first 5 products which have the minimum number of Customers with the $limit function ($limit:5). I have used the Products collection to check all the Products and the customers which have bought it.

Q10. (a)

```
pos> db.Products.find({ "Product ID": 11 }, { "Customer.Customer Name": 1, "_id": null })
[
  {
    _id: null,
    Customer: [
      { 'Customer Name': 'Anika George' },
      { 'Customer Name': 'Aliyah Carlson' },
      { 'Customer Name': 'Patrick Sengupta' },
      { 'Customer Name': 'Adelaide Spradley' },
      { 'Customer Name': 'Jess Keller' },
      { 'Customer Name': 'Irvin Maund' },
      { 'Customer Name': 'Jordyn Ivanov' },
      { 'Customer Name': 'Blakely Phillips' },
      { 'Customer Name': 'Maya Deng' },
      { 'Customer Name': 'Helena Yun' },
      { 'Customer Name': 'Finn Guilbert' },
      { 'Customer Name': 'Taara Yamaguchi' },
      { 'Customer Name': 'Helena Menasinkai' },
      { 'Customer Name': 'Sallie Vickers' },
      { 'Customer Name': 'Eunice Torres' }
    ]
  }
]
```

(b) I have considered Product whose Product-ID is 11 to be the Product that has been recalled. I am using the db.Products.find query to find the Customers who have bought those products. The Customer.Customer Name: 1 means that only the Customer Name needs to be included in the results.

Q11 (a)

```
pos> db.CustomerOrders.aggregate([
...   {
...     $unwind: "$Order Placed"
...   },
...   {
...     $unwind: "$Order Placed.Items"
...   },
...   {
...     $group: {
...       _id: "$Customer Name",
...       ProductQuantity: { $sum: "$Order Placed.Items.Quantity" }
...     }
...   },
...   {
...     $match: {
...       ProductQuantity: { $gt: 70 }
...     }
...   }
... ])
[
  { _id: 'Callie Ali', ProductQuantity: 74 },
  { _id: 'Sofia Madson', ProductQuantity: 77 },
  { _id: 'Virginia Cooper', ProductQuantity: 95 },
  { _id: 'Wallace Yates', ProductQuantity: 71 },
  { _id: 'Bennett Parker', ProductQuantity: 71 },
  { _id: 'Jonathan Sanchez', ProductQuantity: 79 },
  { _id: 'Leroy Ahmed', ProductQuantity: 71 },
  { _id: 'Luv Whitehead', ProductQuantity: 85 },
  { _id: 'Floyd Sanderson', ProductQuantity: 76 },
  { _id: 'Landon Sen-Sharma', ProductQuantity: 74 },
  { _id: 'Journee Chandra', ProductQuantity: 74 },
  { _id: 'Lou Gardner', ProductQuantity: 75 }
]
```

(b) Orders which have more than 70 products are considered fraudulent in this scenario. In the above query I have unwinded first the Order Placed array and then the Items array inside of the Order Placed. Then, the sum of all the Product Quantities are taken in the “ProducQuantity”. We then match all the Product Quantities which are greater than 70 (match function is used to filter data). All the Customer Names who have placed such orders will be displayed. I used the CustomerOrders Collection because I wanted to check all the Customers who have placed Orders which have more than 70 product quantity. All these attributes were present in the CustomerOrders collection only.

Q12

(a) I personally felt MongoDB to be easier than SQL. Performing Complex queries as well as aggregation in JSON Document was very easy as JSON enhances readability due to its simple language. Importing these JSON documents in MongoDB could be done by just one command mongoimport. MongoDB was much simpler to use especially for nested data in each of our collections.

(b) In SQL, data could only be stored in a fixed schema. This structure ensures that only valid data types are entered based on the defined schema. We use JOIN query to combine two or more tables and perform complex queries.

In MongoDB, there was flexibility as it was schema-less. Hence, we could any add data in the database. Db.Collection.aggregate query is used to perform several different queries inside it to perform a complex query.

(c) Denormalization is used in database to improve the performance of the database by improving data retrieval speed. Additionally in different Collections, Lets say Customers and CustomerOrder collections, similar data was present, however the context was different for each collection. By normalizing data, there is a potential that the context would be lost.