DBMSL Assignment No 10

Name: Ravindra Dayaram Bagul
Roll No: C31105
Batch:T1

Output

#Data

```
Orders> db.order.find()
[
  {
    _id: ObjectId("652e5ba74b17d3480662a1bb"),
    order_id: 1,
    cust_id: 'A1',
    cust_name: 'ABC',
    phone_no: [ 1234567890, 987654321 ],
    email: 'ABC@gmail.com',
    item: 'Laptop',
    DOR: ISODate("2022-12-31T18:30:00.000Z"),
    qty: 2,
    amt: 10000,
    status: 'P'
  },
  {
    _id: ObjectId("652e5c574b17d3480662a1bc"),
    order_id: 2,
    cust_id: 'A2',
    cust_name: 'XYZ',
    phone_no: 1234567890,
    email: 'XYZ@gmail.com',
    item: 'TV',
    DOR: ISODate("2023-02-12T18:30:00.000Z"),
    qty: 1,
    amt: 20000,
    status: 'D'
  }
]
```

1. **A) Create a simple index on cust_id and also create a simple index on Item_name.**

```
Orders> db.order.ensureIndex({'cust_id':1})
[ 'cust_id_1' ]
Orders> db.order.ensureIndex({'item':1})
[ 'item_1' ]
Orders> db.order.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { cust_id: 1 }, name: 'cust_id_1' },
  { v: 2, key: { item: 1 }, name: 'item_1' }
]
```

B)Try to make a duplicate entry.

```
Orders>
db.order.insertOne({order_id:2,cust_id:'A2',cust_name:'XYZ',phone_no:1234567890
,email:'XYZ@gmail.com',item:'TV',DOR:new Date('2023-2-
13'),qty:2,amt:20000,status:'P'})
{
  acknowledged: true,
  insertedId: ObjectId("652e60d44b17d3480662a1bd")
}
```

```
Orders> db.order.find({cust_id:'A2'})
      [
        {
            _id: ObjectId("652e5c574b17d3480662a1bc"),
            order_id: 2,
            cust_id: 'A2',
            cust_name: 'XYZ',
            phone_no: 1234567890,
            email: 'XYZ@gmail.com',
            item: 'TV',
            DOR: ISODate("2023-02-12T18:30:00.000Z"),
            qty: 1,
            amt: 20000,
            status: 'D'
        },
        {
            _id: ObjectId("652e60d44b17d3480662a1bd"),
            order_id: 2,
            cust_id: 'A2',
            cust_name: 'XYZ',
            phone_no: 1234567890,
            email: 'XYZ@gmail.com',
            item: 'TV',
            DOR: ISODate("2023-02-12T18:30:00.000Z"),
            qty: 2,
            amt: 20000,
            status: 'P'
        }
      ]
```

2. A)Create unique index on the order_id key.

```
Orders> db.order.ensureIndex({order_id:1},{unique:1})
      [ 'order_id_1' ]
Orders> db.order.getIndexes()
      [
        { v: 2, key: { _id: 1 }, name: '_id_' },
        { v: 2, key: { cust_id: 1 }, name: 'cust_id_1' },
        { v: 2, key: { item: 1 }, name: 'item_1' },
        { v: 2, key: { order_id: 1 }, name: 'order_id_1',
          unique: true }
      ]
```

B)Try to make duplicate entry.

```
Orders>
db.order.insertOne({order_id:2,cust_id:'A2',cust_name:'XYZ',phone_no:1234567890
,email:'XYZ@gmail.com',item:'TV',DOR:new                    Date('2023-2-
13'),qty:2,amt:20000,status:'P'})

MongoServerError: E11000 duplicate key error collection:
Orders.order index: order_id_1 dup key: { order_id: 2 }
```

3. A)Create a multikey index on phone_no.

```
Orders> db.order.ensureIndex({phone_no:1},{multi:1})
      [ 'phone_no_1' ]
Orders> db.order.getIndexes()
      [
        { v: 2, key: { _id: 1 }, name: '_id_' },
        { v: 2, key: { cust_id: 1 }, name: 'cust_id_1' },
        { v: 2, key: { item: 1 }, name: 'item_1' },
```

```
                    { v: 2, key: { order_id: 1 }, name: 'order_id_1',
                      unique: true },
                    { v: 2, key: { phone_no: 1 }, name: 'phone_no_1' }
                  ]
```

B)Find the customers with 2 phone nos.

```
Orders> db.order.find({phone_no:{$size:2}})
      [
        {
            _id: ObjectId("652e5ba74b17d3480662a1bb"),
            order_id: 1,
            cust_id: 'A1',
            cust_name: 'ABC',
            phone_no: [ 1234567890, 987654321 ],
            email: 'ABC@gmail.com',
            item: 'Laptop',
            DOR: ISODate("2022-12-31T18:30:00.000Z"),
            qty: 2,
            amt: 10000,
            status: 'P'
        }
      ]
```

4. A)Create a sparse index on email_id key and show the effects without indexing.

```
Order> db.order.find({email:'ABC@gmail.com'}).explain()
      {
        explainVersion: '2',
        queryPlanner: {
            namespace: 'Order.order',
            indexFilterSet: false,
            parsedQuery: { email: { '$eq': 'ABC@gmail.com' } },
            queryHash: 'B9CE814D',
            planCacheKey: 'A757FE25',
            maxIndexedOrSolutionsReached: false,
            maxIndexedAndSolutionsReached: false,
            maxScansToExplodeReached: false,
            winningPlan: {
              queryPlan: {
                  stage: 'COLLSCAN',
                  planNodeId: 1,
                  filter: { email: { '$eq':'ABC@gmail.com' } },
                  direction: 'forward'
              },
              slotBasedPlan: {
                  slots: '$$RESULT=s5 env: { s3 = 1697544085775
(NOW), s7 = "ABC@gmail.com", s1 =
TimeZoneDatabase(Africa/Khartoum...America/Indiana/Knox) (timeZoneDB), s2 =
Nothing (SEARCH_META) }',
                  stages: '[1] filter {traverseF(s4,
lambda(l1.0) { ((l1.0 == s7) ?: false) }, false)} \n' +
                    '[1] scan s5 s6 none none none none
lowPriority [s4 = email] @"e58977ea-171d-48eb-8876-6e86f61146ad" true false '
              }
            },
            rejectedPlans: []
        },
        command: { find: 'order', filter: { email:
'ABC@gmail.com' }, '$db': 'Order' },
        serverInfo: {
            host: 'DESKTOP-DC8IT0V',
            port: 27017,
```

```
                            version: '7.0.2',
                            gitVersion:
'02b3c655e1302209ef046da6ba3ef6749dd0b62a'
                        },
                        serverParameters: {
                            internalQueryFacetBufferSizeBytes: 104857600,
                            internalQueryFacetMaxOutputDocSizeBytes: 104857600,

      internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
                            internalDocumentSourceGroupMaxMemoryBytes:
104857600,
                            internalQueryMaxBlockingSortMemoryUsageBytes:
104857600,
                            internalQueryProhibitBlockingMergeOnMongoS: 0,
                            internalQueryMaxAddToSetBytes: 104857600,

      internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
                            internalQueryFrameworkControl: 'trySbeEngine'
                        },
                        ok: 1
                    }
```

B)Create the sparse index.

```
Orders> db.order.ensureIndex({email:1},{sparse:1})
      [ 'email_1' ]
Orders> db.order.getIndexes()
      [
        { v: 2, key: { _id: 1 }, name: '_id_' },
        { v: 2, key: { cust_id: 1 }, name: 'cust_id_1' },
        { v: 2, key: { item: 1 }, name: 'item_1' },
        { v: 2, key: { order_id: 1 }, name: 'order_id_1',
            unique: true },
        { v: 2, key: { phone_no: 1 }, name: 'phone_no_1' },
        { v: 2, key: { email: 1 }, name: 'email_1', sparse:
            true }
      ]
```

C)show the effects with indexing.

```
Orders> db.order.find({email:'ABC@gmail.com'}).explain()
      {
        explainVersion: '2',
        queryPlanner: {
            namespace: 'Orders.order',
            indexFilterSet: false,
            parsedQuery: { email: { '$eq': 'ABC@gmail.com' } },
            queryHash: 'B9CE814D',
            planCacheKey: '674E89D1',
            maxIndexedOrSolutionsReached: false,
            maxIndexedAndSolutionsReached: false,
            maxScansToExplodeReached: false,
            winningPlan: {
              queryPlan: {
                  stage: 'FETCH',
                  planNodeId: 2,
                  inputStage: {
                    stage: 'IXSCAN',
                    planNodeId: 1,
                    keyPattern: { email: 1 },
                    indexName: 'email_1',
                    isMultiKey: false,
                    multiKeyPaths: { email: [] },
                    isUnique: false,
```

```
                                    isSparse: true,
                                    isPartial: false,
                                    indexVersion: 2,
                                    direction: 'forward',
                                    indexBounds: { email: [ '["ABC@gmail.com",
"ABC@gmail.com"]' ] }
                                }
                            },
                            slotBasedPlan: {
                                slots: '$$RESULT=s11 env: { s3 =
1697544343321 (NOW), s6 = KS(3C41424340676D61696C2E636F6D00FE04), s1 =
TimeZoneDatabase(Africa/Khartoum...America/Indiana/Knox) (timeZoneDB), s2 =
Nothing (SEARCH_META), s5 = KS(3C41424340676D61696C2E636F6D000104), s10 =
{"email" : 1} }',
                                stages: '[2] nlj inner [] [s4, s7, s8, s9,
s10] \n' +
                                  '    left \n' +
                                  '        [1] cfilter {(exists(s5) &&
exists(s6))} \n' +
                                  '        [1] ixseek s5 s6 s9 s4 s7 s8 []
@"79cf34d4-e449-4210-a944-1b2e36eb936a" @"email_1" true \n' +
                                  '    right \n' +
                                  '        [2] limit 1 \n' +
                                  '        [2] seek s4 s11 s12 s7 s8 s9 s10
[] @"79cf34d4-e449-4210-a944-1b2e36eb936a" true false \n'
                                }
                            },
                            rejectedPlans: []
                        },
                        command: {
                            find: 'order',
                            filter: { email: 'ABC@gmail.com' },
                            '$db': 'Orders'
                        },
                        serverInfo: {
                            host: 'DESKTOP-DC8IT0V',
                            port: 27017,
                            version: '7.0.2',
                            gitVersion:
'02b3c655e1302209ef046da6ba3ef6749dd0b62a'
                        },
                        serverParameters: {
                            internalQueryFacetBufferSizeBytes: 104857600,
                            internalQueryFacetMaxOutputDocSizeBytes: 104857600,

      internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
                            internalDocumentSourceGroupMaxMemoryBytes:
104857600,
                            internalQueryMaxBlockingSortMemoryUsageBytes:
104857600,
                            internalQueryProhibitBlockingMergeOnMongoS: 0,
                            internalQueryMaxAddToSetBytes: 104857600,

      internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
                            internalQueryFrameworkControl: 'trySbeEngine'
                        },
                        ok: 1
                    }
```

5. A)Display all indexes created on order collection.

```
Orders> db.order.getIndexes()
      [
        { v: 2, key: { _id: 1 }, name: '_id_' },
```

```
                        { v: 2, key: { cust_id: 1 }, name: 'cust_id_1' },
                        { v: 2, key: { item: 1 }, name: 'item_1' },
                        { v: 2, key: { order_id: 1 }, name: 'order_id_1',
unique: true },
                        { v: 2, key: { phone_no: 1 }, name: 'phone_no_1' },
                        { v: 2, key: { email: 1 }, name: 'email_1', sparse:
true }
                    ]
```

B)Also show the size of indexes.

```
Orders> db.order.totalIndexSize()
      188416
```

6. Delete all indexes.

```
Orders> db.order.dropIndexes()
      {
        nIndexesWas: 6,
        msg: 'non-_id indexes dropped for collection',
        ok: 1
      }
      Orders> db.order.getIndexes()
      [ { v: 2, key: { _id: 1 }, name: '_id_' } ]
```

7. A) Find Total no of orders received so far

```
Orders> db.order.find().count()
      3
```

B) How many orders are pending.

```
Orders> db.order.find({status:'P'}).count()
      2
```

#Inserting Data

```
Orders>
db.order.insertOne({order_id:4,cust_id:'A4',cust_name:'LMN',phone_no:[123456789
0.2345678910],item:'Microwave',DOR:new Date('2023-3-
20'),qty:6,amt:20000,status:'D'})
      {
        acknowledged: true,
        insertedId: ObjectId("652e7b88afcf0cbe9e7187a8")
      }
Orders>
db.order.insertOne({order_id:5,cust_id:'A4',cust_name:'LMN',phone_no:[123456789
0.2345678910],item:'TV',DOR:new Date('2023-4-20'),qty:4,amt:50000,status:'D'})
      {
        acknowledged: true,
        insertedId: ObjectId("652e7bdaafcf0cbe9e7187a9")
      }
```

8. Display all customer names of orders collection with no repetition.

```
Orders> db.order.distinct('cust_name')
      [ 'ABC', 'LMN', 'XYZ' ]
```

9. Show results and details of sorting documents based on amount.

```
Orders> db.order.find().sort({'amt':1})
      [
        {
          _id: ObjectId("652e5ba74b17d3480662a1bb"),
```

```
        order_id: 1,
        cust_id: 'A1',
        cust_name: 'ABC',
        phone_no: [ 1234567890, 987654321 ],
        email: 'ABC@gmail.com',
        item: 'Laptop',
        DOR: ISODate("2022-12-31T18:30:00.000Z"),
        qty: 2,
        amt: 10000,
        status: 'P'
    },
    {
        _id: ObjectId("652e5c574b17d3480662a1bc"),
        order_id: 3,
        cust_id: 'A2',
        cust_name: 'XYZ',
        phone_no: 1234567890,
        email: 'XYZ@gmail.com',
        item: 'TV',
        DOR: ISODate("2023-02-12T18:30:00.000Z"),
        qty: 1,
        amt: 20000,
        status: 'D'
    },
    {
        _id: ObjectId("652e60d44b17d3480662a1bd"),
        order_id: 2,
        cust_id: 'A2',
        cust_name: 'XYZ',
        phone_no: 1234567890,
        email: 'XYZ@gmail.com',
        item: 'TV',
        DOR: ISODate("2023-02-12T18:30:00.000Z"),
        qty: 2,
        amt: 20000,
        status: 'P'
    },
    {
        _id: ObjectId("652e7b88afcf0cbe9e7187a8"),
        order_id: 4,
        cust_id: 'A4',
        cust_name: 'LMN',
        phone_no: [ 1234567890.2345679 ],
        item: 'Microwave',
        DOR: ISODate("2023-03-19T18:30:00.000Z"),
        qty: 6,
        amt: 20000,
        status: 'D'
    },
    {
        _id: ObjectId("652e7bdaafcf0cbe9e7187a9"),
        order_id: 5,
        cust_id: 'A4',
        cust_name: 'LMN',
        phone_no: [ 1234567890.2345679 ],
        item: 'TV',
        DOR: ISODate("2023-04-19T18:30:00.000Z"),
        qty: 4,
        amt: 50000,
        status: 'D'
    }
]
```

10. Show how many orders are placed by each customer.

```
Orders>
db.order.aggregate({$group:{_id:'$cust_name',Count:{$sum:1}}})
[
  { _id: 'XYZ', Count: 2 },
  { _id: 'ABC', Count: 1 },
  { _id: 'LMN', Count: 2 }
]
```

11. Display all customer ids and their total pending order amount in descending order.

```
Orders>
db.order.aggregate({$match:{status:'P'}},{$group:{_id:'$cust_id',sum_p:{$sum:'$
amt'}}},{$sort:{sum_p:-1}},{$limit:1})
[ { _id: 'A2', sum_p: 20000 } ]
```

12. Display all customer ids in ascending order with total order amount which have been is delivered.

```
Orders>
db.order.aggregate({$match:{status:'D'}},{$group:{_id:'$cust_id',sum_p:{$sum:'$
amt'}}},{$sort:{sum_p:1}})
[ { _id: 'A2', sum_p: 20000 }, { _id: 'A4', sum_p: 70000 } ]
```

13. Show top three Selling Items from orders collection.

```
Orders>
db.order.aggregate({$group:{_id:'$item',sum_item:{$sum:'$qty'}}},{$sort:{sum_it
em:-1}},{$limit:3})
[
  { _id: 'TV', sum_item: 7 },
  { _id: 'Microwave', sum_item: 6 },
  { _id: 'Laptop', sum_item: 2 }
]
```

14. Find the date on which maximum orders are received.

```
Orders>
db.order.aggregate({$group:{_id:'$DOR',count_order:{$sum:1}}},{$sort:{count_ord
er:-1}},{$limit:1})
[ { _id: ISODate("2023-02-12T18:30:00.000Z"), count_order: 2 }
]
```

15. Find which customer has placed maximum orders.

```
Orders>
db.order.aggregate({$group:{_id:'$cust_name',count_orderid:{$sum:1}}},{$sort:{c
ount_orderid:-1}},{$limit:1})
[ { _id: 'XYZ', count_orderid: 2 } ]
```