**Name – Shahane Akash Dilip Roll No – C32260 Class – TE Div:2**

**Batch – T8 Subject - DBMS**

**Assignment No – B2**

**Title:** Implement aggregation and indexing with suitable example using MongoDB

**Problem Statement:** Create an orders collection with keys order\_id, cust\_id, cust\_name, phone\_no(array field), email\_id(optional field), item\_name, DtOfOrder, quantity, amount, status(P :pending/D:delivered)

**----------------------------------------------------------------**

**Queries –**

**1. Create a simple index on cust\_id and also create a simple index on Item\_name. Try to make a duplicate entry.**

b2> db.orders.createIndex({ cust\_id: 1 })

cust\_id\_1

b2> db.orders.createIndex({ item\_name: 1 })

item\_name\_1

b2> db.orders.insertOne({

... order\_id: 2,

... cust\_id: 101, // This cust\_id already exists

... cust\_name: "Jane Doe",

... phone\_no: ["5555555555"],

... email\_id: "jane.doe@example.com",

... item\_name: "Product B",

... DtOfOrder: new Date("2023-10-18T12:00:00Z"),

... quantity: 1,

... amount: 50.25,

... status: "P"

... })

{

acknowledged: true,

insertedId: ObjectId("652dad7386073509c15f76d2")

}

b2> db.orders.getIndexes()

[

{ v: 2, key: { \_id: 1 }, name: '\_id\_' },

{ v: 2, key: { cust\_id: 1 }, name: 'cust\_id\_1' },

{ v: 2, key: { item\_name: 1 }, name: 'item\_name\_1' }

]

**2. Create unique index on the order\_id key and try to make duplicate entry.**

b2> db.orders.createIndex({ order\_id: 1 }, { unique: true })

order\_id\_1

b2> db.orders.insertOne({ order\_id: 2, cust\_id: 101, /\* This cust\_id already exists\*/ cust\_name: "Jane Doe", phone\_no: ["5555555555"], email\_id: "jane.doe@example.com", item\_name: "Product B", DtOfOrder: new Date("2023-10-18T12:00:00Z"), quantity: 1, amount: 50.25, status: "P" })

MongoServerError: E11000 duplicate key error collection: b2.orders index: order\_id\_1 dup key: { order\_id: 2 }

**3. Create a multikey index on phone\_no and find the customers with 2 phone numbers.**

b2> db.orders.createIndex({ phone\_no: 1 }, { multikey: true })

phone\_no\_1

b2> db.orders.find({ phone\_no: { $size: 2 } })

[

{

\_id: ObjectId("652dacdb86073509c15f76d1"),

order\_id: 1,

cust\_id: 101,

cust\_name: 'John Doe',

phone\_no: [ '1234567890', '9876543210' ],

email\_id: 'john.doe@example.com',

item\_name: 'Product A',

DtOfOrder: ISODate("2023-10-17T12:00:00.000Z"),

quantity: 2,

amount: 100.5,

status: 'P'

}

]

**4. Create a sparse index on email\_id key and show the effects with and without indexing.**

b2> db.orders.find({ email\_id: { $exists: true } }).explain()

{

explainVersion: '2',

queryPlanner: {

namespace: 'b2.orders',

indexFilterSet: false,

parsedQuery: { email\_id: { '$exists': true } },

queryHash: '3A720C84',

planCacheKey: '3B778558',

maxIndexedOrSolutionsReached: false,

maxIndexedAndSolutionsReached: false,

maxScansToExplodeReached: false,

winningPlan: {

queryPlan: {

stage: 'COLLSCAN',

planNodeId: 1,

filter: { email\_id: { '$exists': true } },

direction: 'forward'

},

slotBasedPlan: {

slots: '$$RESULT=s5 env: { s3 = 1697492884775 (NOW), s2 = Nothing (SEARCH\_META), s1 = TimeZoneDatabase(America/Detroit...Europe/Vienna) (timeZoneDB) }',

stages: '[1] filter {\n' +

' let [\n' +

' l1.0 = s4 \n' +

' ] \n' +

' in exists(l1.0) \n' +

'} \n' +

'[1] scan s5 s6 none none none none lowPriority [s4 = email\_id] @"e67fe1d2-b9e7-4251-87e1-02ecc3abef67" true false '

}

},

rejectedPlans: []

},

command: {

find: 'orders',

filter: { email\_id: { '$exists': true } },

'$db': 'b2'

},

serverInfo: {

host: 'R2-D2',

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

}

b2> db.orders.createIndex({ email\_id: 1 }, { sparse: true })

email\_id\_1

b2> db.orders.find({ email\_id: { $exists: true } }).explain()

{

explainVersion: '2',

queryPlanner: {

namespace: 'b2.orders',

indexFilterSet: false,

parsedQuery: { email\_id: { '$exists': true } },

queryHash: '3A720C84',

planCacheKey: 'F8FF47D7',

maxIndexedOrSolutionsReached: false,

maxIndexedAndSolutionsReached: false,

maxScansToExplodeReached: false,

winningPlan: {

queryPlan: {

stage: 'FETCH',

planNodeId: 2,

inputStage: {

stage: 'IXSCAN',

planNodeId: 1,

keyPattern: { email\_id: 1 },

indexName: 'email\_id\_1',

isMultiKey: false,

multiKeyPaths: { email\_id: [] },

isUnique: false,

isSparse: true,

isPartial: false,

indexVersion: 2,

direction: 'forward',

indexBounds: { email\_id: [ '[MinKey, MaxKey]' ] }

}

},

slotBasedPlan: {

slots: '$$RESULT=s9 env: { s2 = Nothing (SEARCH\_META), s8 = {"email\_id" : 1}, s3 = 1697493008036 (NOW), s1 = TimeZoneDatabase(America/Detroit...Europe/Vienna) (timeZoneDB) }',

stages: '[2] nlj inner [] [s4, s5, s6, s7, s8] \n' +

' left \n' +

' [1] ixseek KS(0A0104) KS(F0FE04) s7 s4 s5 s6 [] @"e67fe1d2-b9e7-4251-87e1-02ecc3abef67" @"email\_id\_1" true \n' +

' right \n' +

' [2] limit 1 \n' +

' [2] seek s4 s9 s10 s5 s6 s7 s8 [] @"e67fe1d2-b9e7-4251-87e1-02ecc3abef67" true false \n'

}

},

rejectedPlans: []

},

command: {

find: 'orders',

filter: { email\_id: { '$exists': true } },

'$db': 'b2'

},

serverInfo: {

host: 'R2-D2',

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. Display all indexes created on order collection and Also show the size of indexes.**

b2> db.orders.getIndexes()

[

{ v: 2, key: { \_id: 1 }, name: '\_id\_' },

{ v: 2, key: { cust\_id: 1 }, name: 'cust\_id\_1' },

{ v: 2, key: { item\_name: 1 }, name: 'item\_name\_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\_id: 1 }, name: 'email\_id\_1', sparse: true }

]

b2> db.orders.totalIndexSize()

188416

**6. Delete all indexes**

b2> db.orders.dropIndexes()

{

nIndexesWas: 6,

msg: 'non-\_id indexes dropped for collection',

ok: 1

}

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

b2> db.orders.find({Status:'D'}).count()

1

**B) How many orders are pending.**

b2> db.orders.find({Status:'P'}).count()

2

**8. Show results and details of sorting documents based on amount**

b2> db.orders.distinct("cust\_name")

[ 'Jane Doe', 'John Doe' ]

**9. A) Find Total no of orders received so far**

b2> db.orders.find({Status:'D'}).count()

1

**B) How many orders are pending.**

b2> db.orders.find({Status:'P'}).count()

3

**10. Show results and details of sorting documents based on amount**

b2> db.orders.find().sort({Amt:1}).pretty()

[

{

\_id: ObjectId("652dacdb86073509c15f76d1"),

order\_id: 1,

cust\_id: 101,

cust\_name: 'John Doe',

phone\_no: [ '1234567890', '9876543210' ],

email\_id: 'john.doe@example.com',

item\_name: 'Product A',

DtOfOrder: ISODate("2023-10-17T12:00:00.000Z"),

quantity: 2,

amount: 100.5,

status: 'P'

},

{

\_id: ObjectId("652dad7386073509c15f76d2"),

order\_id: 2,

cust\_id: 101,

cust\_name: 'Jane Doe',

phone\_no: [ '5555555555' ],

email\_id: 'jane.doe@example.com',

item\_name: 'Product B',

DtOfOrder: ISODate("2023-10-18T12:00:00.000Z"),

quantity: 1,

amount: 50.25,

status: 'P'

},

{

\_id: ObjectId("652db3d686073509c15f76d4"),

order\_id: 2,

cust\_id: 102,

cust\_name: 'Jane Doe',

phone\_no: [ '555-555-5555' ],

email\_id: 'jane.doe@example.com',

item\_name: 'Product B',

DtOfOrder: ISODate("2023-10-18T12:00:00.000Z"),

quantity: 1,

amount: 50.25,

status: 'P'

},

{

\_id: ObjectId("652db3d686073509c15f76d5"),

order\_id: 3,

cust\_id: 103,

cust\_name: 'Bob Smith',

phone\_no: [ '111-111-1111', '222-222-2222' ],

item\_name: 'Product C',

DtOfOrder: ISODate("2023-10-19T14:30:00.000Z"),

quantity: 3,

amount: 150.75,

status: 'D'

}]

**11. Show how many orders are placed by each customer.**

b2>db.orders.aggregate({$group:{\_id:"$Cust\_name",cnt\_of\_order:{$sum:1}}})

[ { \_id: null, cnt\_of\_order: 6 } ]

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

b2> db.orders.aggregate([

... {$match: { status: "P" },{$group: {\_id: "$cust\_id",

totalPendingAmount: { $sum: "$amount" }}},{$sort: { totalPendingAmount: 1 }}])

[

{ \_id: 101, totalPendingAmount: 150.75 },

{ \_id: 104, totalPendingAmount: 100.5 },

{ \_id: 102, totalPendingAmount: 50.25 }

]

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

b2> db.orders.aggregate([{$match: { status: "D" } },{$group: {\_id: "$cust\_id",totalOrderAmount: { $sum: "$amount" }}},{$$sort: { \_id: 1 }}])

[

{ \_id: 103, totalOrderAmount: 150.75 },

{ \_id: 105, totalOrderAmount: 120 }

]

**14. Show top three Selling Items from orders collection**

b2> db.orders.aggregate({$group:{\_id:"$Item\_name",totqty:{$sum:"$Qty"}}}, {$sort:{totqty:-1}},{$limit:3})

[

{ "\_id": "Product B", "totalQuantity": 9 },

{ "\_id": "Product A", "totalQuantity": 7 },

{ "\_id": "Product C", "totalQuantity": 4 }

]

**15. Find the date on which maximum orders are received.**

b2> db.orders.aggregate([{ $group: { \_id: "$DtOfOrder", orderCount: { $sum: 1 } } }, { $sort: { orderCount: -1 } }, { $l$limit: 1 }])

[ { \_id: ISODate("2023-10-18T12:00:00.000Z"), orderCount: 2 } ]

**16. Find which customer has placed maximum orders.**

b2> db.orders.aggregate([{ $group: { \_id: "$cust\_id", orderCount: { $sum: 1 } } }, { $sort: { orderCount: -1 } }, { $lim$limit: 1 }])

[ { \_id: 101, orderCount: 2 } ]