# ECS789P - SEMI-STRUCTURED DATA AND ADVANCED DATA MODELLING - 2021/22

## MongoDB Design and Implementation

## GROUP 21: POST-GRADUATE

Sanjay Ramesh : 210811700

Ritika Gupta : 210487158

Aditya Ronak Shah: 210841431

Bineeta Kachhap : 210619025

## DESCRIPTION:

The document specification is to the design and implement a mongodb database for an airline company. The system is required to store information about the flights it runs, journey bookings, passengers and air crew. The document has been prepared as part of coursework-2 for ECS789P Semi-Structured Data and Advanced Data Modelling - 2021/22

## ASSUMPTIONS:

* 6 document collection has been created as Employees, Customers, Bookings, Flight Schedule, Employee Code, Planes. The details have been mentioned in the Schema section.
* Each collection has a primary identification key. For e.g., Employees collection has employee\_id, Bookings has booking\_id etc.
* Four types of employees are considered for this coursework: Pilots, Cabin Crew, Maintenance Staff, Booking Clerk.
* For the purpose of project and as experiment the day wise salary and hours of work has been added as 1: N embedded array within the employee detail.
* For this course the reference fields are not added in the validator schema.
* A standard salary has been maintained per employee type.
* All employees and customer have one address each.
* The flight schedules and travel plan are considered for 16th and 17th November 2021. For this course work, there are only two days that the airline was active for the month of November.
* All payments are done by Card.
* All the booking done by customer were done on 1st November 2021.
* Each plane has maximum capacity of 10. Flying range for the planes were measured in km.
* All the date time fields are in ISO format.
* Each Airport has a cost that comprises of its maintenance, fuel cost, cost for flights standby. For this project, the maintenance cost is born at the departure city for flights. Each flight at the time of departure halts for 2 hours each.

## SOLUTION/DELIVERABLES:

Following files are part of deliverable that specifies the coursework.

* Gr21\_210811700\_210487158\_210841431\_210619025\_Report.pdf :
* Group21\_schemaValidator\_cw2.js
* Group21\_dbsetup\_cw2.js
* Group21\_queries\_cw2.js
* Group21\_profiler\_cw2.js
* Group21\_explain\_indexes\_cw2.js
* Data file: employees\_CW2.json , bookings\_CW2.json, customers\_CW2.json, flightSchedule\_CW2.json, employeeCode\_CW2.json, planes\_CW2.json.

The data files were shared among the group member for loading the data. The pdf file contains most of the query executed, in case of an issues the .js files are provided as well.

## SCHEMA:

Following collection were created.

##### Employees:

Table

Description automatically generatedTable

Description automatically generatedTable

Description automatically generatedThe collection contains the list of current and old employees, along with their contact details. The field joining date and last working date are of in ISO format and contains employment details. For current employees the last working date is empty. To showcase the understanding of 1:N embedded array, the field work schedule has been created. The work schedule contains details of work per day and salary received per day. The work time is calculated in hours.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Comment** |
| employee\_id\* | string | custom id for each employees. |
| firstName\* | string |  |
| lastName\* | string |  |
| email\* | string |  |
| contactNumber\* | double |  |
| address\* | object | 1:1 embedded array |
| joiningDate\* | date |  |
| lastWorkingDate\* | date |  |
| employeeType\* | object | 1:1 embedded array |
| employeeWorkSchedule\* | array[1] | 1:N embedded object |

##### Customers:

The collection has list of customers, along with the contact details. The address is 1:1 array with building No, streets city and zip code stored in it.

|  |  |  |
| --- | --- | --- |
| **Field** | Type | Comments |
| customer\_id\* | string | custom id for each customer |
| title\* | string |  |
| firstName\* | string |  |
| lastName\* | string |  |
| email | string |  |
| contactNumber\* | double |  |
| address\* | object | 1:1 embedded array |

##### Bookings:

The booking collection stores the details of each booking done. The collection contains the customer who made the booking along with the flight details. The payment has been done by card and payment amount for each booking done by the customer.

|  |  |  |
| --- | --- | --- |
| **Field** | Type | Comments |
| book\_id\* | string | custom id for each booking |
| customer\_id\* | string | reference to customer\_id from Customer collections |
| flight\_id\* | array | list of reference flight\_id from Flight Schedule collection |
| paymentMode\* | string | only card |
| bookingDate\* | date | has been taken as 1st November 2021 |
| bookingAmount\* | double |  |

##### Airports:

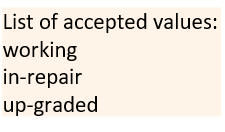
Table

Description automatically generatedThe Airport contains the list of functioning aiport available in various cities, the airportCost contains details about the maintenance cost, refuel and hourly cost.

|  |  |  |
| --- | --- | --- |
| **Field** | type | Comment |
| cityID | string |  |
| city | string |  |
| airportCost | object | 1:1 embedded array contains details about maintenance cost, refuel and hourly cost |

##### Planes:

Table

Description automatically generated The Planes collection has details of all the planes that are available.

|  |  |  |
| --- | --- | --- |
| **Field** | Type | Comments |
| plane\_id\* | string | custom id for each type of planes |
| make\* | string |  |
| model\* | string |  |
| serviceTime | double |  |
| state\* | object | 1:1 embedded array |
| capacity\* | double | each has capacity of 10 |
| flyingRange\* | double |  |
| unit\* | string | accepted values of km, miles |

##### Employee code:

The employee code collection stored the type of Employees available in the airline. The collection has been created to store the details of each employee type and salaries associated for the post.

|  |  |  |
| --- | --- | --- |
| **Field** | Type | Comments |
| code\_id\* | string | custom\_id for each employee type |
| employeeType\* | string | Pilots, Cabin Crew, Maintenance Staff, Booking Clerk |
| standardSalary\* | double |  |

##### 

##### Flight Schedule:

The flight schedule collection stores the list of all the flights that are schedules to fly. Each flight has details of the plane being used and the departure and arrival. time and city.

|  |  |  |
| --- | --- | --- |
| **Field** | Type | Comments |
| flight\_id\* | string | custom id for each flight schedules |
| plane\_id\* | string | reference to plane to be used |
| departure\_city\_id\* | string | departure city |
| arrival\_city\_id\* | string | arrival City |
| departureTime\* | date |  |
| arrivalTime\* | date |  |
| employee\_list\* | array[3] |  |

The below image outlines the complete schema and structure of the database:

A picture containing text, receipt, screenshot

Description automatically generated

## SCHEMA VALIDATOR FOR EACH COLLECTION

1. Schema validator for EmployeeCode Collection:

db.createCollection("employeeCode", {

validator: {

$jsonSchema: {

bsonType: "object",

required: [ "code\_id", "employeeType", "standardSalary" ],

properties: {

code\_id: { bsonType: "string" },

employeeType: { bsonType: "string", enum:["Booking Clerks", "Maintenance Staff", "Pilot", "Cabin Crew"]},

standardSalary: { bsonType: "double" }

}

}

}})

1. Schema validator for Planes Collection:

db.createCollection("planes", {

validator: {

$jsonSchema: {

bsonType: "object",

required: [ "plane\_id", "make", "model", "state", "capacity", "flyingRange", "unit"],

properties: {

plane\_id: { bsonType: "string" },

make: { bsonType: "string" },

model: { bsonType: "string" },

serviceTime: { bsonType: "double" },

state: {

bsonType: "object",

required: [ "status"],

properties:{

status: { bsonType: "string", enum : ["working","in-repair","upgraded"] },

repairCost: { bsonType: "double" }

}

},

capacity: { bsonType: "double" },

flyingRange: { bsonType: "double" },

unit: { enum : ["km","miles"], bsonType: "string" }

}

}

}})

1. Schema validator for Bookings Collection:

db.createCollection("bookings", {

validator: {

$jsonSchema: {

bsonType: "object",

required: [ "book\_id", "flight\_id", "customer\_id", "paymentMode", "bookingDate", "bookingAmount"],

properties: {

book\_id: { bsonType: "string" },

flight\_id: { bsonType: "array", minItems: 1, items: { bsonType: "string" }},

customer\_id: { bsonType: "string" },

paymentMode: { bsonType: "string" },

bookingDate: { bsonType: "date" },

bookingAmount: { bsonType: "double" }

}

}

}})

1. Schema validator for Customers Collection:

db.createCollection("customers", {

validator: {

$jsonSchema: {

bsonType: "object",

required: [ "customer\_id", "firstName", "lastName", "contactNumber","address"],

properties: {

customer\_id: { bsonType: "string" },

title: { bsonType: "string" },

firstName: { bsonType: "string" },

lastName: { bsonType: "string" },

email: { bsonType: "string" },

contactNumber: { bsonType: "double" },

address: {

bsonType: "object",

required: [ "buildingNo", "street", "city" ,"zipcode"],

properties:{

buildingNo: { bsonType: "double" },

street: { bsonType: "string" },

city: { bsonType: "string" },

zipcode: { bsonType: "string" }

}

}

}

}} })

1. 5. Schema for Airports Collection

db.createCollection("airports", {

validator: {

$jsonSchema: {

bsonType: "object",

required: [ "city\_id", "city", "airportCost"],

properties: {

city\_id: { bsonType: "string" },

city: { bsonType: "string" },

airportCost: {

bsonType: "object",

required: [ "refuelCost", "maintainanceCost", "hourlyStopRate"],

properties:{

refuelCost: { bsonType: "double" },

maintainanceCost: { bsonType: "double" },

hourlyStopRate: { bsonType: "double" }

}

}

}

}

}

})

1. Schema validator for FlightSchedule Collection:

db.createCollection("flightSchedule", {

validator: {

$jsonSchema: {

bsonType: "object",

required: [ "flight\_id", "plane\_id", "departureCity", "arrivalCity", "departureTime", "arrivalTime","employee\_list"],

properties: {

flight\_id: { bsonType: "string" },

plane\_id: { bsonType: "string" },

departureCity: { bsonType: "string" },

arrivalCity: { bsonType: "string" },

departureTime: { bsonType: "date" },

arrivalTime: { bsonType: "date" },

employee\_list: {

bsonType: "array",

minItems: 3,

items: { bsonType: "string" }

}

}

}} })

1. Schema validator for Employees Collection:

db.createCollection("employees", {

validator: {

$jsonSchema: {

bsonType: "object",

required: [ "employee\_id", "firstName", "lastName", "email", "contactNumber","address","joiningDate", "employeeType"],

properties: {

employee\_id: { bsonType: "string" },

firstName: { bsonType: "string" },

lastName: { bsonType: "string" },

email: { bsonType: "string" },

contactNumber: { bsonType: "double" },

address: {

bsonType: "object",

required: [ "buildingNo", "street", "city" ,"zipcode"],

properties:{

buildingNo: { bsonType: "double" },

street: { bsonType: "string" },

city: { bsonType: "string" },

zipcode: { bsonType: "string" }

}

},

joiningDate: { bsonType: "date" },

lastWorkingDate: { bsonType: "date" },

employeeWorkSchedule: {

bsonType: "array",

items: {

bsonType: "object",

required: [ "date", "hours”, “salary"],

properties:{

date: { bsonType: "date" },

hours: { bsonType: "double" },

salary: { bsonType: "double" }

}

}

},

employeeType: {

bsonType: "object",

required: ["eType"],

properties:{

eType: { bsonType: "string" },

fitToFly: { bsonType: "bool" }

}

}

}

}} })

## SET OF 12 QUERIES

Following set of queries have been written to showcase understanding about the queries and commands. Concepts and mongodb expressions were covered in the below queries: Screenshot with few details have been attached for the queries.

- showcase of aggregate and find utility method

- use of aggregate pipeline with $project, $match, $group, $unwind, $sort, $limit, $lookup.

- usage of utility methods line $gt, $eq, $concat, $where, $year, $month, $dayOfMonth, $sum, $subtract, $exist, $toString, $dateToString, $addToSet, $size, $isArray

1. List of existing Employees name and contact number:

Query by checking if last working day fields is empty.

db.employees.aggregate(

{$match:{"lastWorkingDate":{$exists:false}}},

{$project: {

"\_id":0,

name: {$concat: [ "$firstName", " ", "$lastName"]},

number: "$contactNumber",

address: {$concat: [ {$toString:"$address.buildingNo"}, " ", "$address.street",", ","$address.city", "-", "$address.zipcode"]}}}

)

Text

Description automatically generated

1. Customers with doctoral degree:

Query using find method and checking the title as Dr.

db.customers.find({title:"Dr."},{customer\_id:true, firstName:true, lastName:true, email:true}).sort({firstName:1}).limit(10)

A screen shot of a computer

Description automatically generated with low confidence

1. Booking ids with more than one flight.

Query by checking the flight list is more than 1

db.bookings.find({$where: "this.flight\_id.length > 1"}).forEach(function(booking){

print("Booking Ids: "+ booking.book\_id+ ", "+"Flight List: "+ booking.flight\_id);

});

Text

Description automatically generated

1. Available flight for dates in November (16/17)

Query by fetching all flight schedule and group by date.

db.flightSchedule.aggregate(

{ $project: {

year: {$year: "$departureTime"},

month: {$month: "$departureTime"},

dayOfMonth: {$dayOfMonth: "$departureTime"},

departureCity: "$departureCity",

arrivalCity: "$arrivalCity"

}},

{ $group: {

\_id: { date: {$concat: [ {$toString:"$year"}, "-", {$toString:"$month"},"-", {$toString:"$dayOfMonth"}] }},

flights: { $push: { $concat: [ "$departureCity", " - ", "$arrivalCity",]}}}})

Text

Description automatically generated

1. Top 5 employee with highest salary.

Done by aggregating employees collection and fetching all data for employeeWorkSchedule and adding up the employeeWorkSchedule.salary field.

db.employees.aggregate(

{$match: {employeeWorkSchedule: {$exists: true}}},

{ $project: {

name : {$concat:["$firstName"," ","$lastName"]},

employeeWorkSchedule: 1 }},

{$unwind: "$employeeWorkSchedule"},

{ $group: {

\_id: "$name", salary: {$sum:"$employeeWorkSchedule.salary"}}},

{ $sort: {"salary": -1}},

{ $limit: 5}

)

Text

Description automatically generated

1. List of employees group by city.

Done by aggregating employees collection and fetching all data and grouping by address.city.

db.employees.aggregate(

{$match: {address: {$exists: true}}},

{ $project: {

name : {$concat:["$firstName"," ","$lastName"]},

address: 1 }},

{ $group: {

\_id: "$address.city", employeeList : {$push:"$name"} }}

)

Text

Description automatically generated

1. Top 10 oldest employees by joining date(The find query has been written to do further for Indexes and explain utility method.)

Done by find method with applying sort to the joining date, the find method has been used for further analysis of indexes. The aggregate method matched all data and sorts by joining date.

db.employees.find({"employee\_id":{$exists:false}},{employee\_id:true, firstName:true, lastName:true, joiningDate:true}).sort({joiningDate:1}).limit(10)

or

db.employees.aggregate(

{$match: {joiningDate: {$exists: true}}},

{ $project: {

name : {$concat:["$firstName"," ","$lastName"]},

joiningDate: 1 }},

{ $sort: {"joiningDate": 1}},

{ $limit: 10})

Text

Description automatically generated

1. Customers who travelled on 16/11/2021

Used aggregate method to match all data in flight schedule collection for 16th Nov and then looked up bookings collection to match all the flight and booking\_ids and customer\_id. With the result of customer\_id looked up customers collection to match all the customer.

db.flightSchedule.aggregate({

$project: {

year: {$year: "$departureTime"},

month: {$month: "$departureTime"},

dayOfMonth: {$dayOfMonth: "$departureTime"},

flight: "$flight\_id"

}},

{ $match:{$and: [{"year":{$eq:2021}}, {"month":{$eq:11}}, {"dayOfMonth":{$eq:16}} ]}},

{ $group: {

\_id: { date: {$concat: [ {$toString:"$year"}, "-", {$toString:"$month"},"-", {$toString:"$dayOfMonth"}] }},

flights: { $push: "$flight"}}},

{ $unwind: "$flights"},

{ $lookup: {from: "bookings", localField: "flights", foreignField: "flight\_id", as: "bookings"}},

{ $unwind: "$bookings"},

{ $group: { \_id: "$bookings.customer\_id" }},

{ $lookup: {from: "customers", localField: "\_id", foreignField: "customer\_id", as: "customers"}},

{ $unwind: "$customers"},

{

$project: {

\_id: "$\_id",

customer: {$concat:["$customers.title"," ","$customers.firstName"," ","$customers.lastName"]}}})

Text

Description automatically generated

1. Total Earning of the November Month.

Used aggregate method to sum up all the payment done for the month of November from booking collection and sum up all the salary given out from the customer collection and sum up any ongoing maintenance and repair work. The profit was calculated as total payment - salary given - maintenance work.

db.bookings.aggregate(

{ $group:{

\_id: "$bookingDate",

customerPayment: {$sum: "$bookingAmount"}

}},

{ $lookup: {from: "employees", pipeline: [{$match: {employeeWorkSchedule: {$exists: true}}}], as: "employees"} },

{ $unwind: "$employees" },

{ $group:{

\_id: "$customerPayment",

totalSalary:{ $sum: {$sum:"$employees.employeeWorkSchedule.salary"}}}},

{ $project:{

\_id: "Profit",

profitEarned: {$subtract:["$\_id","$totalSalary"]}}},

{ $lookup: {from: "planes", pipeline: [{$match: {"state.status":{$in:["in-repair","upgraded"]}}}], as: "maintainanceCost"}},

{ $unwind: "$maintainanceCost" },

{ $group:{

\_id: "$profitEarned",

maintainanceCost:{$sum:"$maintainanceCost.state.repairCost"}}},

{ $project:{

\_id: "$profitEarned",

profitEarned: {$subtract:["$\_id","$maintainanceCost"]}

}},

{ $lookup: {from: "flightSchedule", pipeline: [{$match: {departureCity: {$exists: true}}}], as: "flightSchedule"}},

{ $unwind: "$flightSchedule" },

{ $lookup: {from: "airports", localField: "flightSchedule.departureCity", foreignField: "city\_id", as: "airport"}},

{ $unwind: "$airport"},

{ $group: {

\_id: "$profitEarned",

maintainanceCost: {$sum: {$add:["$airport.airportCost.refuelCost", "$airport.airportCost.maintainanceCost", {$multiply:["$airport.airportCost.hourlyStopRate",2]} ]}} }},

{ $project:{

\_id: "Profit",

profitEarned: {$subtract:["$\_id","$maintainanceCost"]} }}

)

Text

Description automatically generated

1. Employees working on 17/11/2021

Use aggregate method to match all flight schedules and list of employee id associated to the flight for 17th November. Looked up the employee collection to match all the employee\_id from the employees collection.

db.flightSchedule.aggregate({

$project: {

year: {$year: "$departureTime"},

month: {$month: "$departureTime"},

dayOfMonth: {$dayOfMonth: "$departureTime"},

flight: "$flight\_id",

employee\_list: "$employee\_list"

}},{

$match:{ $and: [{"year":{$eq:2021}},{"month":{$eq:11}},{"dayOfMonth":{$eq:17}} ]}},

{ $unwind: "$employee\_list" },

{

$group: {

\_id: { $concat: [ {$toString:"$year"}, "-", {$toString:"$month"},"-", {$toString:"$dayOfMonth"}] },

employee: { $addToSet: "$employee\_list"}}},

{ $unwind: "$employee"},

{ $lookup: {from: "employees", localField: "employee", foreignField: "employee\_id", as: "employees"}},

{ $unwind: "$employees" },

{$project: {"\_id":"$\_id", Name: {$concat: [ "$employees.firstName", " ", "$employees.lastName"]}, Number: "$employees.contactNumber"}})

Text

Description automatically generated

1. Passenger List who travelled more than one flight.

From the booking collection get all the flight counts per booking, for any count which was greater than 1, the customer\_ids were added to a set and looked up customers collection to get the customer details.

db.bookings.aggregate(

{ $project: {

book\_id: 1,

customer\_id: 1,

flightCount: { $cond: { if: { $isArray: "$flight\_id" }, then: { $size: "$flight\_id" }, else: "NA"} }}},

{ $project :{

book\_id: 1,

customer\_id: 1,

flightCount: 1,

isMore: { $cond: {if: {$gt:["$flightCount",1] }, then: true, else: false}} }},

{ $group: {

\_id: "$isMore", customer\_id: { $addToSet: "$customer\_id"}}},

{ $match: { \_id: true}},

{ $lookup: {from: "customers", localField: "customer\_id", foreignField: "customer\_id", as: "customers"}},

{ $unwind: "$customers"},

{$project: {

\_id: "Flights",

customer: {$concat:["$customers.firstName"," ","$customers.lastName"]}

}}

)

Text

Description automatically generated

1. Top 5 customers with highest expenditure.

Used aggregate method to sort all the bookings by the payment/booking amount, the top 5 list of customers were looked up in the customers collection for details.

db.bookings.aggregate(

{ $project: {

book\_id: 1,

customer\_id: 1,

bookingAmount: 1}},

{ $sort: {"bookingAmount": -1}},

{ $limit: 5},

{ $lookup: {from: "customers", localField: "customer\_id", foreignField: "customer\_id", as: "customers"}},

{ $unwind: "$customers"},

{$project: {

\_id: "Top5",

customer: {$concat:["$customers.firstName"," ","$customers.lastName"]},

address: {$concat: [ {$toString:"$customers.address.buildingNo"}, " ", "$customers.address.street",", ","$customers.address.city", "-", "$customers.address.zipcode"]}}}

)

Text

Description automatically generated

1. Travel history of each passenger with date, price, flight and travel time details.

Used aggregate method to project all the customers, from the bookings collection mapped all the flight schedule and from the flight schedule collection mapped all the travel time, destination and arrival details. The details are shows together with $project.

db.customers.aggregate(

{ $project: {

name : {$concat:["$title"," ","$firstName"," ","$lastName"]},

customer\_id: 1 }},

{ $lookup: {from: "bookings", localField: "customer\_id", foreignField: "customer\_id", as: "bookings"}},

{ $unwind: "$bookings" },

{ $project: {

name : "$name",

customer\_id: 1,

flight: "$bookings.flight\_id",

price: "$bookings.bookingAmount"}},

{ $unwind: "$flight"},

{ $lookup: {from: "flightSchedule", localField: "flight", foreignField: "flight\_id", as: "schedule"}},

{ $unwind: "$schedule"},

{ $project: {

\_id: 0,

name : 1,

price: 1,

travel: {$concat:["$schedule.departureCity","-","$schedule.arrivalCity"]},

travelTime: {$concat:[{ $dateToString: { format: "%Y:%m:%d-%H:%M:%S", date: "$schedule.departureTime"}}," - ",{ $dateToString: { format: "%Y:%m:%d-%H:%M:%S", date: "$schedule.arrivalTime"}}]}

}}

)

Text

Description automatically generated

## PROFILER:

The profiler details are shown in the **Group21\_profiler\_cw2.js** as the result output contents were very huge.

## EXPLAIN AND INDEXES:

Query number 7, 3, 2 are used as an example to showcase the analysis done on usage of indexes. As customers, bookings, employees collection had maximum amount of dummy data the indexes were created on this collection for their respective custom ids.

Query :7

#### 1.. Explain and indexes for employees Collection on employee\_id field. The below output has been produced for query no. 7 (Top 10 oldest employees by joining date)

#### Output a) without indexes:

- The highlighted "stage" : "COLLSCAN" shows that full column scan was done on the document collection to fetch all relevant data.

- The field "executionTimeMillisEstimate" : 0 shows that it took less than 0 milli seconds to execute the whole query.

- The "executionSuccess" : true shows that there are no logic or formatting issue and the query executed successfully.

db.employees.find({"employee\_id":{$exists:false}},{employee\_id:true, firstName:true, lastName:true, joiningDate:true}).sort({joiningDate:1}).limit(10).explain("executionStats")

{

"explainVersion" : "1",

"queryPlanner" : {

"namespace" : "qmulairline.employees",

"indexFilterSet" : false,

"parsedQuery" : {

"employee\_id" : {

"$not" : {

"$exists" : true

}

}

},

"maxIndexedOrSolutionsReached" : false,

"maxIndexedAndSolutionsReached" : false,

"maxScansToExplodeReached" : false,

"winningPlan" : {

"stage" : "PROJECTION\_SIMPLE",

"transformBy" : {

"employee\_id" : true,

"firstName" : true,

"lastName" : true,

"joiningDate" : true

},

"inputStage" : {

"stage" : "SORT",

"sortPattern" : {

"joiningDate" : 1

},

"memLimit" : 104857600,

"limitAmount" : 10,

"type" : "simple",

"inputStage" : {

"stage" : "COLLSCAN",

"filter" : {

"employee\_id" : {

"$not" : {

"$exists" : true

}

}

},

"direction" : "forward"

}

}

},

"rejectedPlans" : [ ]

},

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 0,

"executionTimeMillis" : 0,

"totalKeysExamined" : 0,

"totalDocsExamined" : 21,

"executionStages" : {

"stage" : "PROJECTION\_SIMPLE",

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 24,

"advanced" : 0,

"needTime" : 23,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"transformBy" : {

"employee\_id" : true,

"firstName" : true,

"lastName" : true,

"joiningDate" : true

},

"inputStage" : {

"stage" : "SORT",

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 24,

"advanced" : 0,

"needTime" : 23,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"sortPattern" : {

"joiningDate" : 1

},

"memLimit" : 104857600,

"limitAmount" : 10,

"type" : "simple",

"totalDataSizeSorted" : 0,

"usedDisk" : false,

"inputStage" : {

"stage" : "COLLSCAN",

"filter" : {

"employee\_id" : {

"$not" : {

"$exists" : true

}

}

},

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 23,

"advanced" : 0,

"needTime" : 22,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"direction" : "forward",

"docsExamined" : 21

}

}

}

},

"command" : {

"find" : "employees",

"filter" : {

"employee\_id" : {

"$exists" : false

}

},

"limit" : 10,

"singleBatch" : false,

"sort" : {

"joiningDate" : 1

},

"projection" : {

"employee\_id" : true,

"firstName" : true,

"lastName" : true,

"joiningDate" : true

},

"$db" : "qmulairline"

},

"serverInfo" : {

"host" : "DESKTOP-O2C1LRU",

"port" : 27017,

"version" : "5.0.4",

"gitVersion" : "62a84ede3cc9a334e8bc82160714df71e7d3a29e"

},

"serverParameters" : {

"internalQueryFacetBufferSizeBytes" : 104857600,

"internalQueryFacetMaxOutputDocSizeBytes" : 104857600,

"internalLookupStageIntermediateDocumentMaxSizeBytes" : 104857600,

"internalDocumentSourceGroupMaxMemoryBytes" : 104857600,

"internalQueryMaxBlockingSortMemoryUsageBytes" : 104857600,

"internalQueryProhibitBlockingMergeOnMongoS" : 0,

"internalQueryMaxAddToSetBytes" : 104857600,

"internalDocumentSourceSetWindowFieldsMaxMemoryBytes" : 104857600

},

"ok" : 1

}

#### Created indexes for customer collection for customer\_id field.

db.employees.getIndexes();

db.employees.createIndex({"employee\_id ":1});

db.employees.dropIndex("employee\_id\_1");

#### Output b) after indexes:

- The highlighted "stage" : "IXSCAN" shows that index column scan was done on the document collection to fetch all relevant data.

- The field "executionTimeMillisEstimate" : 0 shows that it took less than 0 milli seconds to execute the whole query.

- The "executionSuccess" : true shows that there are no logic or formatting issue and the query executed successfully.

db.employees.find({"employee\_id":{$exists:false}},{employee\_id:true, firstName:true, lastName:true, joiningDate:true}).sort({joiningDate:1}).limit(10).explain("executionStats")

{

"explainVersion" : "1",

"queryPlanner" : {

"namespace" : "qmulairline.employees",

"indexFilterSet" : false,

"parsedQuery" : {

"employee\_id" : {

"$not" : {

"$exists" : true

}

}

},

"maxIndexedOrSolutionsReached" : false,

"maxIndexedAndSolutionsReached" : false,

"maxScansToExplodeReached" : false,

"winningPlan" : {

"stage" : "PROJECTION\_SIMPLE",

"transformBy" : {

"employee\_id" : true,

"firstName" : true,

"lastName" : true,

"joiningDate" : true

},

"inputStage" : {

"stage" : "SORT",

"sortPattern" : {

"joiningDate" : 1

},

"memLimit" : 104857600,

"limitAmount" : 10,

"type" : "simple",

"inputStage" : {

"stage" : "FETCH",

"filter" : {

"employee\_id" : {

"$not" : {

"$exists" : true

}

}

},

"inputStage" : {

"stage" : "IXSCAN",

"keyPattern" : {

"employee\_id" : 1

},

"indexName" : "employee\_id\_1",

"isMultiKey" : false,

"multiKeyPaths" : {

"employee\_id" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"employee\_id" : [

"[null, null]"

]

}

}

}

}

},

"rejectedPlans" : [ ]

},

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 0,

"executionTimeMillis" : 0,

"totalKeysExamined" : 0,

"totalDocsExamined" : 0,

"executionStages" : {

"stage" : "PROJECTION\_SIMPLE",

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 2,

"advanced" : 0,

"needTime" : 1,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"transformBy" : {

"employee\_id" : true,

"firstName" : true,

"lastName" : true,

"joiningDate" : true

},

"inputStage" : {

"stage" : "SORT",

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 2,

"advanced" : 0,

"needTime" : 1,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"sortPattern" : {

"joiningDate" : 1

},

"memLimit" : 104857600,

"limitAmount" : 10,

"type" : "simple",

"totalDataSizeSorted" : 0,

"usedDisk" : false,

"inputStage" : {

"stage" : "FETCH",

"filter" : {

"employee\_id" : {

"$not" : {

"$exists" : true

}

}

},

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 1,

"advanced" : 0,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"docsExamined" : 0,

"alreadyHasObj" : 0,

"inputStage" : {

"stage" : "IXSCAN",

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 1,

"advanced" : 0,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"keyPattern" : {

"employee\_id" : 1

},

"indexName" : "employee\_id\_1",

"isMultiKey" : false,

"multiKeyPaths" : {

"employee\_id" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"employee\_id" : [

"[null, null]"

]

},

"keysExamined" : 0,

"seeks" : 1,

"dupsTested" : 0,

"dupsDropped" : 0

}

}

}

}

},

"command" : {

"find" : "employees",

"filter" : {

"employee\_id" : {

"$exists" : false

}

},

"limit" : 10,

"singleBatch" : false,

"sort" : {

"joiningDate" : 1

},

"projection" : {

"employee\_id" : true,

"firstName" : true,

"lastName" : true,

"joiningDate" : true

},

"$db" : "qmulairline"

},

"serverInfo" : {

"host" : "DESKTOP-O2C1LRU",

"port" : 27017,

"version" : "5.0.4",

"gitVersion" : "62a84ede3cc9a334e8bc82160714df71e7d3a29e"

},

"serverParameters" : {

"internalQueryFacetBufferSizeBytes" : 104857600,

"internalQueryFacetMaxOutputDocSizeBytes" : 104857600,

"internalLookupStageIntermediateDocumentMaxSizeBytes" : 104857600,

"internalDocumentSourceGroupMaxMemoryBytes" : 104857600,

"internalQueryMaxBlockingSortMemoryUsageBytes" : 104857600,

"internalQueryProhibitBlockingMergeOnMongoS" : 0,

"internalQueryMaxAddToSetBytes" : 104857600,

"internalDocumentSourceSetWindowFieldsMaxMemoryBytes" : 104857600

},

"ok" : 1

}

>

>

Query:3

#### 2. Explain and indexes for bookings collection on flight\_id field. The below output has been produced for query no. 3.(Booking ids with more than one flight.)

#### Output a) without indexes:

- The highlighted "stage" : "COLLSCAN" shows that full column scan was done on the document collection to fetch all relevant data.

- The field "executionTimeMillisEstimate" : 27 shows that it took 27 milli seconds to execute the whole query.

- The "executionSuccess" : true shows that there are no logic or formatting issue and the query executed successfully.

- In the executionStats, for executionStages COLLSCAN was performed, and it returned "nReturned" : 8 data based on the filter condition provided in the find query.

> db.bookings.find({ $and: [{"flight\_id":{$exists:true}}, {$where: "this.flight\_id.length > 1"}]}).explain("executionStats")

{

"explainVersion" : "1",

"queryPlanner" : {

"namespace" : "qmulairline.bookings",

"indexFilterSet" : false,

"parsedQuery" : {

"$where" : {

"code" : "this.flight\_id.length > 1"

}

},

"maxIndexedOrSolutionsReached" : false,

"maxIndexedAndSolutionsReached" : false,

"maxScansToExplodeReached" : false,

"winningPlan" : {

"stage" : "COLLSCAN",

"filter" : {

"$where" : {

"code" : "this.flight\_id.length > 1"

}

},

"direction" : "forward"

},

"rejectedPlans" : [ ]

},

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 8,

"executionTimeMillis" : 33,

"totalKeysExamined" : 0,

"totalDocsExamined" : 23,

"executionStages" : {

"stage" : "COLLSCAN",

"filter" : {

"$where" : {

"code" : "this.flight\_id.length > 1"

}

},

"nReturned" : 8,

"executionTimeMillisEstimate" : 27,

"works" : 25,

"advanced" : 8,

"needTime" : 16,

"needYield" : 0,

"saveState" : 1,

"restoreState" : 1,

"isEOF" : 1,

"direction" : "forward",

"docsExamined" : 23

}

},

"command" : {

"find" : "bookings",

"filter" : {

"$where" : "this.flight\_id.length > 1"

},

"$db" : "qmulairline"

},

"serverInfo" : {

"host" : "DESKTOP-O2C1LRU",

"port" : 27017,

"version" : "5.0.4",

"gitVersion" : "62a84ede3cc9a334e8bc82160714df71e7d3a29e"

},

"serverParameters" : {

"internalQueryFacetBufferSizeBytes" : 104857600,

"internalQueryFacetMaxOutputDocSizeBytes" : 104857600,

"internalLookupStageIntermediateDocumentMaxSizeBytes" : 104857600,

"internalDocumentSourceGroupMaxMemoryBytes" : 104857600,

"internalQueryMaxBlockingSortMemoryUsageBytes" : 104857600,

"internalQueryProhibitBlockingMergeOnMongoS" : 0,

"internalQueryMaxAddToSetBytes" : 104857600,

"internalDocumentSourceSetWindowFieldsMaxMemoryBytes" : 104857600

},

"ok" : 1

}

>

#### Created indexes for bookings collection for flight\_id field.

db.bookings.getIndexes();

db.bookings.createIndex({"flight\_id":1});

db.bookings.dropIndex("flight\_id\_1");

#### Output b) after indexes:

- The highlighted "stage" : "IXSCAN" shows that index column scan was done on the document collection to fetch all relevant data.

- The field "executionTimeMillisEstimate" : 4 shows that it took 4 milli seconds to execute the whole query.

There was as significant decrease in the execution time after the indexes as compared to Output A which was "executionTimeMillisEstimate" : 27.

- The "executionSuccess" : true shows that there are no logic or formatting issue and the query executed successfully.

- In the executionStats, for executionStages IXSCAN was performed, and it returned "nReturned" : 23 data

as there are overall 23 documents in bookings collection and then in FETCH stage it returned "nReturned" : 8 data based on the filter condition provided in the find query.

> db.bookings.find({ $and: [{"flight\_id":{$exists:true}}, {$where: "this.flight\_id.length > 1"}]}).explain("executionStats")

{

"explainVersion" : "1",

"queryPlanner" : {

"namespace" : "qmulairline.bookings",

"indexFilterSet" : false,

"parsedQuery" : {

"$and" : [

{

"flight\_id" : {

"$exists" : true

}

},

{

"$where" : {

"code" : "this.flight\_id.length > 1"

}

}

]

},

"maxIndexedOrSolutionsReached" : false,

"maxIndexedAndSolutionsReached" : false,

"maxScansToExplodeReached" : false,

"winningPlan" : {

"stage" : "FETCH",

"filter" : {

"$and" : [

{

"flight\_id" : {

"$exists" : true

}

},

{

"$where" : {

"code" : "this.flight\_id.length > 1"

}

}

]

},

"inputStage" : {

"stage" : "IXSCAN",

"keyPattern" : {

"flight\_id" : 1

},

"indexName" : "flight\_id\_1",

"isMultiKey" : true,

"multiKeyPaths" : {

"flight\_id" : [

"flight\_id"

]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"flight\_id" : [

"[MinKey, MaxKey]"

]

}

}

},

"rejectedPlans" : [ ]

},

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 8,

"executionTimeMillis" : 101,

"totalKeysExamined" : 33,

"totalDocsExamined" : 23,

"executionStages" : {

"stage" : "FETCH",

"filter" : {

"$and" : [

{

"flight\_id" : {

"$exists" : true

}

},

{

"$where" : {

"code" : "this.flight\_id.length > 1"

}

}

]

},

"nReturned" : 8,

"executionTimeMillisEstimate" : 4,

"works" : 34,

"advanced" : 8,

"needTime" : 25,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"docsExamined" : 23,

"alreadyHasObj" : 0,

"inputStage" : {

"stage" : "IXSCAN",

"nReturned" : 23,

"executionTimeMillisEstimate" : 0,

"works" : 34,

"advanced" : 23,

"needTime" : 10,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"keyPattern" : {

"flight\_id" : 1

},

"indexName" : "flight\_id\_1",

"isMultiKey" : true,

"multiKeyPaths" : {

"flight\_id" : [

"flight\_id"

]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"flight\_id" : [

"[MinKey, MaxKey]"

]

},

"keysExamined" : 33,

"seeks" : 1,

"dupsTested" : 33,

"dupsDropped" : 10

}

}

},

"command" : {

"find" : "bookings",

"filter" : {

"$and" : [

{

"flight\_id" : {

"$exists" : true

}

},

{

"$where" : "this.flight\_id.length > 1"

}

]

},

"$db" : "qmulairline"

},

"serverInfo" : {

"host" : "DESKTOP-O2C1LRU",

"port" : 27017,

"version" : "5.0.4",

"gitVersion" : "62a84ede3cc9a334e8bc82160714df71e7d3a29e"

},

"serverParameters" : {

"internalQueryFacetBufferSizeBytes" : 104857600,

"internalQueryFacetMaxOutputDocSizeBytes" : 104857600,

"internalLookupStageIntermediateDocumentMaxSizeBytes" : 104857600,

"internalDocumentSourceGroupMaxMemoryBytes" : 104857600,

"internalQueryMaxBlockingSortMemoryUsageBytes" : 104857600,

"internalQueryProhibitBlockingMergeOnMongoS" : 0,

"internalQueryMaxAddToSetBytes" : 104857600,

"internalDocumentSourceSetWindowFieldsMaxMemoryBytes" : 104857600

},

"ok" : 1

}

>

Query no:2

#### 3. Explain and indexes for Customer Collection on customer\_id field. The below output has been produced for query no. 2.(Customers with doctoral degree)

#### Output a) without indexes:

- The highlighted "stage" : "COLLSCAN" shows that full column scan was done on the document collection to fetch all relevant data.

- The field "executionTimeMillisEstimate" : 0 shows that it took less than 0 milli seconds to execute the whole query.

- The "executionSuccess" : true shows that there are no logic or formatting issue and the query executed successfully.

- In the executionStats, for executionStages COLLSCAN was performed, and it returned "nReturned" : 8 data based on the filter condition provided in the find query.

> db.customers.find({ $and: [{"customer\_id":{$exists:true}}, {title:"Dr."}]},{customer\_id:true, firstName:true, lastName:true, email:true}).sort({firstName:1}).limit(10).explain("executionStats")

{

"explainVersion" : "1",

"queryPlanner" : {

"namespace" : "qmulairline.customers",

"indexFilterSet" : false,

"parsedQuery" : {

"$and" : [

{

"title" : {

"$eq" : "Dr."

}

},

{

"customer\_id" : {

"$exists" : true

}

}

]

},

"maxIndexedOrSolutionsReached" : false,

"maxIndexedAndSolutionsReached" : false,

"maxScansToExplodeReached" : false,

"winningPlan" : {

"stage" : "PROJECTION\_SIMPLE",

"transformBy" : {

"customer\_id" : true,

"firstName" : true,

"lastName" : true,

"email" : true

},

"inputStage" : {

"stage" : "SORT",

"sortPattern" : {

"firstName" : 1

},

"memLimit" : 104857600,

"limitAmount" : 10,

"type" : "simple",

"inputStage" : {

"stage" : "COLLSCAN",

"filter" : {

"$and" : [

{

"title" : {

"$eq" : "Dr."

}

},

{

"customer\_id" : {

"$exists" : true

}

}

]

},

"direction" : "forward"

}

}

},

"rejectedPlans" : [ ]

},

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 8,

"executionTimeMillis" : 0,

"totalKeysExamined" : 0,

"totalDocsExamined" : 25,

"executionStages" : {

"stage" : "PROJECTION\_SIMPLE",

"nReturned" : 8,

"executionTimeMillisEstimate" : 0,

"works" : 36,

"advanced" : 8,

"needTime" : 27,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"transformBy" : {

"customer\_id" : true,

"firstName" : true,

"lastName" : true,

"email" : true

},

"inputStage" : {

"stage" : "SORT",

"nReturned" : 8,

"executionTimeMillisEstimate" : 0,

"works" : 36,

"advanced" : 8,

"needTime" : 27,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"sortPattern" : {

"firstName" : 1

},

"memLimit" : 104857600,

"limitAmount" : 10,

"type" : "simple",

"totalDataSizeSorted" : 2404,

"usedDisk" : false,

"inputStage" : {

"stage" : "COLLSCAN",

"filter" : {

"$and" : [

{

"title" : {

"$eq" : "Dr."

}

},

{

"customer\_id" : {

"$exists" : true

}

}

]

},

"nReturned" : 8,

"executionTimeMillisEstimate" : 0,

"works" : 27,

"advanced" : 8,

"needTime" : 18,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"direction" : "forward",

"docsExamined" : 25

}

}

}

},

"command" : {

"find" : "customers",

"filter" : {

"$and" : [

{

"customer\_id" : {

"$exists" : true

}

},

{

"title" : "Dr."

}

]

},

"limit" : 10,

"singleBatch" : false,

"sort" : {

"firstName" : 1

},

"projection" : {

"customer\_id" : true,

"firstName" : true,

"lastName" : true,

"email" : true

},

"$db" : "qmulairline"

},

"serverInfo" : {

"host" : "DESKTOP-O2C1LRU",

"port" : 27017,

"version" : "5.0.4",

"gitVersion" : "62a84ede3cc9a334e8bc82160714df71e7d3a29e"

},

"serverParameters" : {

"internalQueryFacetBufferSizeBytes" : 104857600,

"internalQueryFacetMaxOutputDocSizeBytes" : 104857600,

"internalLookupStageIntermediateDocumentMaxSizeBytes" : 104857600,

"internalDocumentSourceGroupMaxMemoryBytes" : 104857600,

"internalQueryMaxBlockingSortMemoryUsageBytes" : 104857600,

"internalQueryProhibitBlockingMergeOnMongoS" : 0,

"internalQueryMaxAddToSetBytes" : 104857600,

"internalDocumentSourceSetWindowFieldsMaxMemoryBytes" : 104857600

},

"ok" : 1

}

>

#### Created indexes for employees collection for employee\_id field.

db.customers.createIndex({"customer\_id ":1});

db.customers.getIndexes();

db.customers.dropIndex("customer\_id \_1");

#### Output b) after indexes:

- The highlighted "stage" : "IXSCAN" shows that index column scan was done on the document collection to fetch all relevant data.

- The field "executionTimeMillisEstimate" : 0 shows that it took less than 0 milli seconds to execute the whole query.

- The "executionSuccess" : true shows that there are no logic or formatting issue and the query executed successfully.

- In the executionStats, for executionStages IXSCAN was performed, and it returned "nReturned" : 25 data

as there are overall 25 documents in bookings collection and then in FETCH stage it returned "nReturned" : 8 data based on the filter condition provided in the find query.

> db.customers.find({ $and: [{"customer\_id":{$exists:true}}, {title:"Dr."}]},{customer\_id:true, firstName:true, lastName:true, email:true}).sort({firstName:1}).limit(10).explain("executionStats")

{

"explainVersion" : "1",

"queryPlanner" : {

"namespace" : "qmulairline.customers",

"indexFilterSet" : false,

"parsedQuery" : {

"$and" : [

{

"title" : {

"$eq" : "Dr."

}

},

{

"customer\_id" : {

"$exists" : true

}

}

]

},

"maxIndexedOrSolutionsReached" : false,

"maxIndexedAndSolutionsReached" : false,

"maxScansToExplodeReached" : false,

"winningPlan" : {

"stage" : "PROJECTION\_SIMPLE",

"transformBy" : {

"customer\_id" : true,

"firstName" : true,

"lastName" : true,

"email" : true

},

"inputStage" : {

"stage" : "SORT",

"sortPattern" : {

"firstName" : 1

},

"memLimit" : 104857600,

"limitAmount" : 10,

"type" : "simple",

"inputStage" : {

"stage" : "FETCH",

"filter" : {

"$and" : [

{

"customer\_id" : {

"$exists" : true

}

},

{

"title" : {

"$eq" : "Dr."

}

}

]

},

"inputStage" : {

"stage" : "IXSCAN",

"keyPattern" : {

"customer\_id" : 1

},

"indexName" : "customer\_id\_1",

"isMultiKey" : false,

"multiKeyPaths" : {

"customer\_id" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"customer\_id" : [

"[MinKey, MaxKey]"

]

}

}

}

}

},

"rejectedPlans" : [ ]

},

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 8,

"executionTimeMillis" : 0,

"totalKeysExamined" : 25,

"totalDocsExamined" : 25,

"executionStages" : {

"stage" : "PROJECTION\_SIMPLE",

"nReturned" : 8,

"executionTimeMillisEstimate" : 0,

"works" : 35,

"advanced" : 8,

"needTime" : 26,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"transformBy" : {

"customer\_id" : true,

"firstName" : true,

"lastName" : true,

"email" : true

},

"inputStage" : {

"stage" : "SORT",

"nReturned" : 8,

"executionTimeMillisEstimate" : 0,

"works" : 35,

"advanced" : 8,

"needTime" : 26,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"sortPattern" : {

"firstName" : 1

},

"memLimit" : 104857600,

"limitAmount" : 10,

"type" : "simple",

"totalDataSizeSorted" : 2404,

"usedDisk" : false,

"inputStage" : {

"stage" : "FETCH",

"filter" : {

"$and" : [

{

"customer\_id" : {

"$exists" : true

}

},

{

"title" : {

"$eq" : "Dr."

}

}

]

},

"nReturned" : 8,

"executionTimeMillisEstimate" : 0,

"works" : 26,

"advanced" : 8,

"needTime" : 17,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"docsExamined" : 25,

"alreadyHasObj" : 0,

"inputStage" : {

"stage" : "IXSCAN",

"nReturned" : 25,

"executionTimeMillisEstimate" : 0,

"works" : 26,

"advanced" : 25,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"keyPattern" : {

"customer\_id" : 1

},

"indexName" : "customer\_id\_1",

"isMultiKey" : false,

"multiKeyPaths" : {

"customer\_id" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"customer\_id" : [

"[MinKey, MaxKey]"

]

},

"keysExamined" : 25,

"seeks" : 1,

"dupsTested" : 0,

"dupsDropped" : 0

}

}

}

}

},

"command" : {

"find" : "customers",

"filter" : {

"$and" : [

{

"customer\_id" : {

"$exists" : true

}

},

{

"title" : "Dr."

}

]

},

"limit" : 10,

"singleBatch" : false,

"sort" : {

"firstName" : 1

},

"projection" : {

"customer\_id" : true,

"firstName" : true,

"lastName" : true,

"email" : true

},

"$db" : "qmulairline"

},

"serverInfo" : {

"host" : "DESKTOP-O2C1LRU",

"port" : 27017,

"version" : "5.0.4",

"gitVersion" : "62a84ede3cc9a334e8bc82160714df71e7d3a29e"

},

"serverParameters" : {

"internalQueryFacetBufferSizeBytes" : 104857600,

"internalQueryFacetMaxOutputDocSizeBytes" : 104857600,

"internalLookupStageIntermediateDocumentMaxSizeBytes" : 104857600,

"internalDocumentSourceGroupMaxMemoryBytes" : 104857600,

"internalQueryMaxBlockingSortMemoryUsageBytes" : 104857600,

"internalQueryProhibitBlockingMergeOnMongoS" : 0,

"internalQueryMaxAddToSetBytes" : 104857600,

"internalDocumentSourceSetWindowFieldsMaxMemoryBytes" : 104857600

},

"ok" : 1

}

>