ECS650/ECS789 Semi-structured Data and Advanced Data Modelling

CW 2: MongoDB Design and Implementation

Group 20: POST_GRADUATE

Sai Krishna Rajesh Chittoor: 210022498

Sri Harsha Musunuru: 210659401

Sai Siddharth Ponugoti: 210011377

Timy Herve: 180582316

Deliverables:

- Group20_Semi_CW2_Report.pdf

- schemamaker.js

- setup.js

- queries.js

- Group20_profiler_output.js

Assumptions:

The tables below are the assumptiomns which we have made for every entry and entity present in the collection accordingly.

Pilots
Every pilot has an ID which is maintained to test the queries correctness, The ID starts from 1. Date of joining and fit for fly are in the ISO format. The dates being used in the fields are in YYYY-MM-DDT00:00:00Z format. This format is intended and used to resemble real life implementation of the system.

Planes
Contains various fields like type of the plane, model name, flying range etc. Every plane make is unique, and their features will be dependingly unique as well

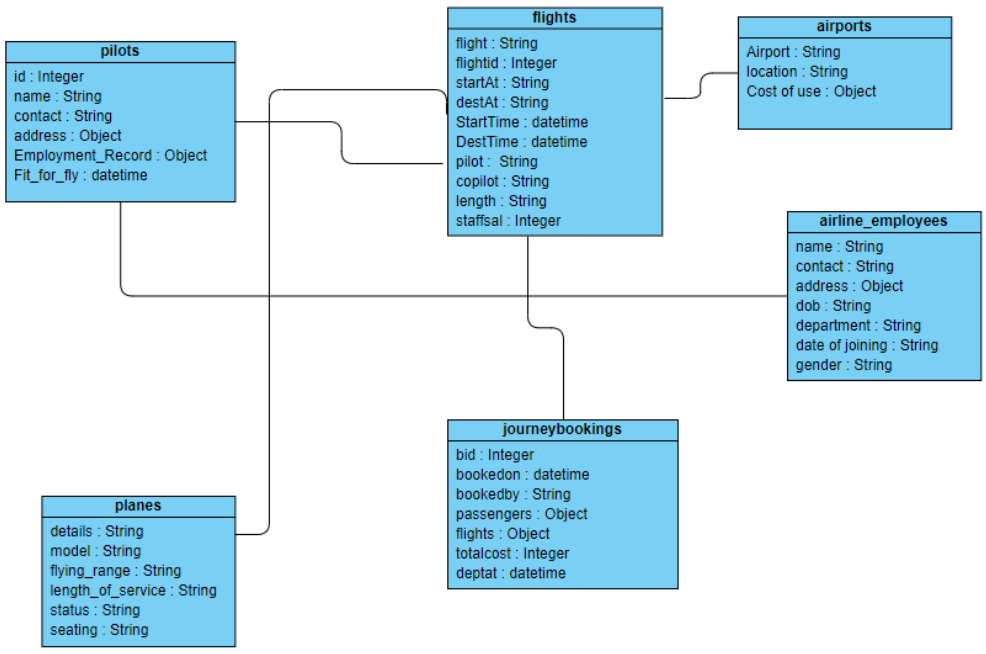
Flights
Every flight has been given flight ID and we have assumed that the specific flight with flight ID will only travel from one source to one destination to avoid further complications. This collection also has details of the start time and end time of the flight as well as the pilots and co-pilots for the flight. We have also added the field staffsal which will store the salary given to the staff of that flight, where staff includes pilots, co-pilots, cabin staff, maintanence staff and booking clerks as well together.

Journey Bookings
This collection will be containing every detail of every booking done. Every booking has a unique booking ID given represented by bid. It also stores details such as name of who did the booking, passenger names and the flight ID of flights used throughout their booked journey. We also store the total cost of the booking under the field totalcost.

Airports
Contains detail of every airport used along with its details such as location and cost to use the airport and its facilities. Cost has been divided as Cost for plane to stop at the airport per hour and cost for its refuelling.

Airline Employees
This collection is where the employees of our airline system are stored. Employee details such as name, address, contact, date of joining , gender and their department are stored here.

Diagram:

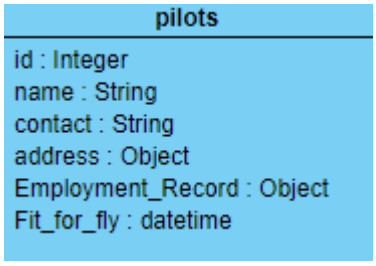


Schema:

Following collections are created.

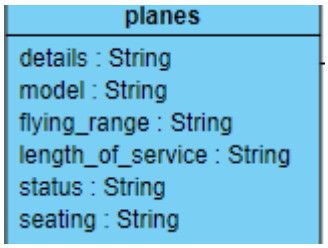
Pilots:

The collection contains list of pilots, their contact, address, employment record and their last fit to fly test.



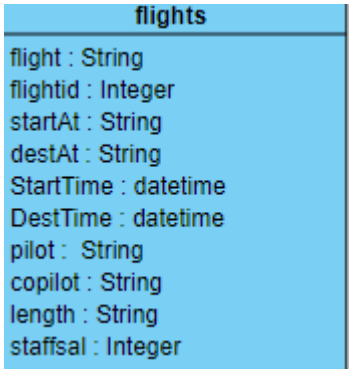
Planes:

The collection contains details,make.flying range,length of service, status and seating of the planes.

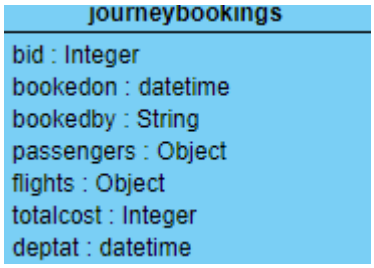


Flights:

The collection contains plane model,flight id , the plane travel details and pilot details.

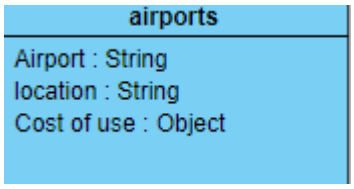


Journeybookings:
The collection contains bookings and booking IDs for all types of bookings.



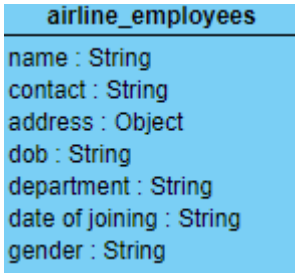
Airports:

The collection contains details of the airport and functionality costs use din airport.



Airline_employees:

The collection contains emplyee details like name, contact,address,dob,department etc..



Schema validator for all the collections:

Schema for pilots Collection:

```
db.createCollection("pilots", {
        validator: {
        $jsonSchema: {
                bsonType: "object",
                required: [ "id", "name", "contact", "address", "Employment_Record", "Fit_for_fly"
],
                properties: {
                        id: { bsonType: "double" },
                        name: { bsonType: "string"},
      contact: { bsonType: "string"},
```

```
        address:{
           bsonType: "object",
           required:["postCode","streetName","civicNo","city"],
           properties: {
              postCode: { bsonType: "String"},
              streetName: { bsonType: "String"},
              civicNo: { bsonType: "Double"},
              city: { bsonType: "String"},
           }

        },
        Employment_Record:{
           bsonType: "object",
           required:["Date_of_join","educational_bg","salary"],
           properties: {
              Date_of_join: { bsonType: "date"},
              educational_bg: { bsonType: "string"},
              salary: { bsonType:"string"}
           }
        },
        Fit_for_fly: {bsonType: "date" }

                    }
          }
          }
})
```

Schema for Planes Collection:

```
db.createCollection("planes", {
          validator: {
          $jsonSchema: {
                    bsonType: "object",
```

```
                required: [ "details","model", "flying_range", "length_of_service",
"status","seating"],
                properties: {
                        details: { bsonType: "string" },
                        model: { bsonType: "string" },
                        flying_range: { bsonType: "string" },
                        length_of_service: { bsonType: "double" },
                        status: { bsonType: "string" },
                        seating: { bsonType: "string" },

        }
        }
        }
})
```

Schema for flights Collection:

```
db.createCollection("flights", {
        validator: {
        $jsonSchema: {
                bsonType: "object",
                required: [ "flight", "flightid", "startAt", "destAt", "startTime",
"DesttTime","pilot","copilot","length","staffsal"],
                properties: {
                        flight: { bsonType: "string" },
                        flightid: { bsonType: "double"},
                        startAt: { bsonType: "string" },
                        destAt: { bsonType: "string" },
                        startTime: { bsonType: "date" },
                        DesttTime: { bsonType: "date" },
        pilot: { bsonType: "string"},
        copilot: { bsonType: "string"},
        length: { bsonType: "string" },
        staffsal: { bsonType: "double"}
```

```
    }
        }
        }
})
```

Schema for journeybookings Collection:

```
db.createCollection("journeybookings", {
        validator: {
        $jsonSchema: {
                bsonType: "object",
                required: [ "bid", "bookedon", "deptat", "bookedby", "passengers",
"flights","totalcost"],
                properties: {
        bid: { bsonType: "double"},
        bookedon: { bsonType: "date" },
        deptat: { bsonType: "date"},
        bookedby: { bsonType: "string"},
        passengers: { bsonType: "object"},
        flights: { bsonType: "object"},
        totalcost: { bsonType: "double"}
    }
    }
        }
        }
)
```

Schema for airports Collection:

```
db.createCollection("airports", {
        validator: {
        $jsonSchema: {
                bsonType: "object",
                required: [ "Airport", "location", "Cost of use"],
                properties: {
```

```
                    Airport: { bsonType: "string" },
                    location: { bsonType: "string"},
                    Cost_of_use: {
            bsonType: "object",
            required:["plane_stop_per_hour","refuelling"],
            properties: {
               plane_stop_per_hour: { bsonType: "double"},
               refuelling: { bsonType: "double"}
            }
         },
      }
          }
          }
})
```

Schema for airline_employees Collection:

```
db.createCollection("airline_employees", {
          validator: {
          $jsonSchema: {
                    bsonType: "object",
                    required:
["name","contact","address","dob","department","salary","doj","gender"],
                    properties: {
                              name: { bsonType: "string"},
          contact: { bsonType: "string"},
          address: {
            bsonType: "object",
            required: [ "postcode","city","street"],
            postcode: { bsonType: "string"},
            city: { bsonType: "string"},
            street: { bsonType: "string"}
```

```
        },
        dob:{ bsonType: "string"},
        department: { bsonType: "string"},
        salary: { bsonType: "double"},
        doj:{ bsonType: "string"},
        gender: { bsonType: "string"},
      }
        }
        }
})
```

Set of 12 Queries:

1.) sort names of pilots

```
db.pilots.aggregate({$sort:{name:1}})
```

O/P

```
[
 {
   _id: ObjectId("61c397569f30a3991ca484ee"),
   id: 4,
   name: 'Alex',
   contact: '09856321457',
   address: {
    postCode: 'SE3 4LB',
    streetName: 'Fed Street',
    civicNo: 73,
    city: 'Liverpool'
   },
   Employment_Record: {
    Date_of_join: ISODate("2005-09-26T01:17:34.000Z"),
    educational_bg: 'bachelors',
    salary: '50,000'
   },
```

```
  Fit_for_fly: ISODate("2004-09-26T01:17:34.000Z")
 },
 {
  _id: ObjectId("61c397569f30a3991ca484ec"),
  id: 2,
  name: 'Jake',
  contact: '07854123692',
  address: {
   postCode: 'E3 4JY',
   streetName: 'eric Street',
   civicNo: 71,
   city: 'London'
  },
  Employment_Record: {
   Date_of_join: ISODate("2003-09-26T01:17:34.000Z"),
   educational_bg: 'bachelors',
   salary: '70,000'
  },
  Fit_for_fly: ISODate("2002-09-26T01:17:34.000Z")
 },
 {
  _id: ObjectId("61c397569f30a3991ca484eb"),
  id: 1,
  name: 'Jean',
  contact: '07894566123',
  address: {
   postCode: 'E3 4LB',
   streetName: 'Upham Street',
   civicNo: 78,
   city: 'London'
  },
```

```
  Employment_Record: {
    Date_of_join: ISODate("2005-09-26T01:17:34.000Z"),
    educational_bg: 'masters',
    salary: '50,000'
  },
  Fit_for_fly: ISODate("2004-09-26T01:17:34.000Z")
},
{
  _id: ObjectId("61c397569f30a3991ca484f1"),
  id: 7,
  name: 'Johnson',
  contact: '07345896524',
  address: {
    postCode: 'NE3 4EB',
    streetName: 'Loki Street',
    civicNo: 34,
    city: 'Manchester'
  },
  Employment_Record: {
    Date_of_join: ISODate("2006-11-26T01:17:34.000Z"),
    educational_bg: 'masters',
    salary: '100,000'
  },
  Fit_for_fly: ISODate("2003-11-27T01:17:34.000Z")
},
{
  _id: ObjectId("61c397569f30a3991ca484ed"),
  id: 3,
  name: 'Johny',
  contact: '05789463125',
  address: {
```

```
    postCode: 'E3 4WR',
    streetName: 'Rhods Street',
    civicNo: 28,
    city: 'London'
   },
   Employment_Record: {
    Date_of_join: ISODate("2010-04-26T01:20:34.000Z"),
    city: 'Belfast'
   },
   Employment_Record: {
    Date_of_join: ISODate("2018-11-26T01:17:34.000Z"),
    educational_bg: 'masters',
    salary: '100,000'
   },
   Fit_for_fly: ISODate("2010-11-27T01:17:34.000Z")
  },
  {
   _id: ObjectId("61c397569f30a3991ca484f0"),
   id: 6,
   name: 'Murphy',
   contact: '09856391457',
   address: {
    postCode: 'N2 4QE',
    streetName: 'Lund Street',
    civicNo: 23,
    city: 'Belfast'
   },
   Employment_Record: {
    Date_of_join: ISODate("2011-09-26T01:17:34.000Z"),
    educational_bg: 'bachelors',
    salary: '50,000'
```

```
  },
  Fit_for_fly: ISODate("2010-09-26T01:17:34.000Z")
 }
]
```

2.) No . of planes working

```
db.planes.find({status:"working"}).count();
```

O/P 4

3.) Pilots whose fit to fly was recent

```
db.pilots.find().sort({Fit_for_fly:-1}).limit(1);
```

O/P

```
[
 {
  _id: ObjectId("61c397569f30a3991ca484ef"),
  id: 5,
  name: 'Mike',
  contact: '07824896524',
  address: {
   postCode: 'W3 4FD',
   streetName: 'Kiako Street',
   civicNo: 35,
   city: 'Belfast'
  },
  Employment_Record: {
   Date_of_join: ISODate("2018-11-26T01:17:34.000Z"),
   educational_bg: 'masters',
   salary: '100,000'
  },
  Fit_for_fly: ISODate("2010-11-27T01:17:34.000Z")
 }
]
```

4.) No. of flights of the same plane model

```
db.flights.aggregate([{ "$group": {_id:"$flight","count":{"$sum":1}}}] )
```

O/P

```
[ { _id: 'Boeing 747-8', count: 2 }, { _id: 'Airbus350', count: 3 } ]
```

5.) Pilots from the same city

```
db.pilots.aggregate(
   {$match: {address: {$exists: true}}},
   { $project: {
   name:1,
   address: 1 }},
   { $group: {
   _id: "$address.city", pilots : {$push:"$name"} }}
   )
```

O/P

```
[
  { _id: 'Liverpool', pilots: [ 'Alex' ] },
  { _id: 'Manchester', pilots: [ 'Johnson' ] },
  { _id: 'Belfast', pilots: [ 'Mike', 'Murphy' ] },
  { _id: 'London', pilots: [ 'Jean', 'Jake', 'Johny' ] }
]
```

6.) Booking IDs for bookings on a specific date

```
db.journeybookings.aggregate({
   $project: {
   year: {$year: "$bookedon"},
   month: {$month: "$bookedon"},
   dayOfMonth: {$dayOfMonth: "$bookedon"},
   flight: "$flights"
   }},
   { $match:{$and: [{"year":{$eq:2021}}, {"month":{$eq:10}}, {"dayOfMonth":{$eq:10}} ]}},
   { $group: {
```

```
  _id: { date: {$concat: [ {$toString:"$year"}, "-", {$toString:"$month"},"-", {$toString:"$dayOfMonth"}] }},
  flights: { $push: "$flight"}}},
  { $unwind: "$flights"},
  { $lookup: {from: "journeybookings", localField: "flights", foreignField: "flights", as: "journeybookings"}},
  { $unwind: "$journeybookings"},
  { $project: { _id: "$journeybookings.bid",name:"$journeybookings.passengers" }})
```

O/P

```
[
 { _id: 33, name: [ 'Emily', 'Roy', 'Jackob' ] },
 { _id: 44, name: [ 'Eric' ] }
]
```

7.) Total salaries given from a department

```
db.airline_employees.aggregate([{ "$group": {_id:"$department",sumSalary: { $sum: "$salary" } } } ,{ $sort: { sumSalary: -1 } } ] )
```

O/P

```
[
 { _id: 'pilots', sumSalary: 53000 },
 { _id: 'cabin staff', sumSalary: 36000 },
 { _id: 'maintenance staff', sumSalary: 32500 },
 { _id: 'booking clerks', sumSalary: 22000 }
]
```

8.) 3 New Latest employees joined

```
db.airline_employees.aggregate(
  {$match: {doj: {$exists: true}}},
  { $project: {
  name : "$name",
  doj: 1 }},
  { $sort: {"doj": -1}},
  { $limit: 3}
```

)

O/P


```
[
 {
  _id: ObjectId("61c3d8f59f30a3991ca484f5"),
  doj: '25-10-2006',
  name: 'phyllis'
 },
 {
  _id: ObjectId("61c3d8f59f30a3991ca484f4"),
  doj: '25-09-2004',
  name: 'pam'
 },
 {
  _id: ObjectId("61c3d8f59f30a3991ca484f3"),
  doj: '25-03-2005',
  name: 'michael'
 }
]
```


9.) Top2 costliest bookings

```
db.journeybookings.aggregate({ $project: {bid: 1,bookedby:1,totalcost:1}},{ $sort: {"totalcost": -1}},{ $limit: 2})
```

O/P


```
[
 {
  _id: ObjectId("61c3dcf99f30a3991ca484fe"),
  bid: 55,
  bookedby: 'kriti',
  totalcost: 1300
 },
 {
```

```
  _id: ObjectId("61c3dcf99f30a3991ca484fb"),
  bid: 22,
  bookedby: 'sam',
  totalcost: 1200
 }
]
```

10.) Male Airline Employee with the highest salary

```
db.airline_employees.find( { gender: "Male"}).sort({salary:-1}).limit(1)
```

O/P

```
[
 {
  _id: ObjectId("61c3d8f59f30a3991ca484f7"),
  name: 'dwight',
  contact: '08500619280',
  address: { postcode: 'e45gh', city: 'london', street: 'nala street' },
  dob: '09-06-1999',
  department: 'pilots',
  salary: 30000,
  doj: '14-06-2004',
  gender: 'Male'
 }
]
```

11.) Pilot with largest salary that joined earlier than the others

```
db.pilots.find({}).sort({"Employment_Record.Date_of_join":1, "Employment_Record.salary":-1}).limit(1)
```

O/P

```
[
 {
  _id: ObjectId("61c397569f30a3991ca484ec"),
  id: 2,
```

```
  name: 'Jake',
  contact: '07854123692',
  address: {
   postCode: 'E3 4JY',
   streetName: 'eric Street',
   civicNo: 71,
   city: 'London'
  },
  Employment_Record: {
   Date_of_join: ISODate("2003-09-26T01:17:34.000Z"),
   educational_bg: 'bachelors',
   salary: '70,000'
  },
  Fit_for_fly: ISODate("2002-09-26T01:17:34.000Z")
 }
]
```

12.) No. of days between booking date and departure date

```
db.journeybookings.aggregate([
  { "$project": {
  "from": "$bookedon",
  "to": "$deptat",
  _id:0,
   "difference": {
    "$divide": [
     { "$subtract": ["$deptat", "$bookedon"] },
     60 * 1000 * 60 * 24
    ]}}
  }])
```

O/P

```
[
```

```
 {
  from: ISODate("2021-12-20T12:00:00.000Z"),
  to: ISODate("2021-12-25T12:00:00.000Z"),
  difference: 5
 },
 {
  from: ISODate("2021-03-18T10:00:00.000Z"),
  to: ISODate("2021-12-20T12:00:00.000Z"),
  difference: 277.0833333333333
 },
 {
  from: ISODate("2021-03-10T14:00:00.000Z"),
  to: ISODate("2021-11-20T12:00:00.000Z"),
  difference: 254.91666666666666
 },
 {
  from: ISODate("2021-10-10T12:00:00.000Z"),
  to: ISODate("2021-12-20T12:00:00.000Z"),
  difference: 71
 },
 {
  from: ISODate("2021-01-08T10:00:00.000Z"),
  to: ISODate("2021-12-20T12:00:00.000Z"),
  difference: 346.0833333333333
 }
]
```

Part 2

Performace tuning and indexing: Before executing performance tuning on some queries like

db.journeybookings.find({$where:"this.bid"}).explain("executionStats")

The performance which we achieved was


```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'AirlineDetails.journeybookings',
    indexFilterSet: false,
    parsedQuery: { '$where': Code("this.bid") },
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'COLLSCAN',
      filter: { '$where': Code("this.bid") },
      direction: 'forward'
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 5,
    executionTimeMillis: 91,
    totalKeysExamined: 0,
    totalDocsExamined: 5,
    executionStages: {
      stage: 'COLLSCAN',
      filter: { '$where': Code("this.bid") },
      nReturned: 5,
      executionTimeMillisEstimate: 3,
      works: 7,
      advanced: 5,
      needTime: 1,
```

```
    needYield: 0,
    saveState: 0,
    restoreState: 0,
    isEOF: 1,
    direction: 'forward',
    docsExamined: 5
   }
 },
 command: {
   find: 'journeybookings',
   filter: { '$where': 'this.bid' },
   '$db': 'AirlineDetails'
 },
 serverInfo: {
   host: 'LAPTOP-93ALV4L6',
   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}
```

As we can see the execution time we achieved for this simple query is 91ms,

We can tune the performance by executing the following commands and using the example function again.

To tune the performance we add a index in our case

db.journeybookings.createIndex({bid”:1}) and then when we execute the command again


```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'AirlineDetails.journeybookings',
    indexFilterSet: false,
    parsedQuery: { '$where': Code("this.bid") },
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'COLLSCAN',
      filter: { '$where': Code("this.bid") },
      direction: 'forward'
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 5,
    executionTimeMillis: 0,
    totalKeysExamined: 0,
    totalDocsExamined: 5,
    executionStages: {
      stage: 'COLLSCAN',
      filter: { '$where': Code("this.bid") },
      nReturned: 5,
```

```
    executionTimeMillisEstimate: 0,
    works: 7,
    advanced: 5,
    needTime: 1,
    needYield: 0,
    saveState: 0,
    restoreState: 0,
    isEOF: 1,
    direction: 'forward',
    docsExamined: 5
   }
 },
 command: {
   find: 'journeybookings',
   filter: { '$where': 'this.bid' },
   '$db': 'AirlineDetails'
 },
 serverInfo: {
   host: 'LAPTOP-93ALV4L6',
   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
}
```

Here as we see the executionTimeMillis has dropped from 91ms to 0ms

Similarly for a second query db.pilots.find({$where:"this.id"}).explain("executionStats")

The performance we received is

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'AirlineDetails.pilots',
    indexFilterSet: false,
    parsedQuery: { '$where': Code("this.id") },
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'COLLSCAN',
      filter: { '$where': Code("this.id") },
      direction: 'forward'
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 7,
    executionTimeMillis: 44,
    totalKeysExamined: 0,
    totalDocsExamined: 7,
    executionStages: {
```

```
    stage: 'COLLSCAN',
    filter: { '$where': Code("this.id") },
    nReturned: 7,
    executionTimeMillisEstimate: 2,
    works: 9,
    advanced: 7,
    needTime: 1,
    needYield: 0,
    saveState: 0,
    restoreState: 0,
    isEOF: 1,
    direction: 'forward',
    docsExamined: 7
  }
},
command: {
  find: 'pilots',
  filter: { '$where': 'this.id' },
  '$db': 'AirlineDetails'
},
serverInfo: {
  host: 'LAPTOP-93ALV4L6',
  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
}
```

After adding index db.pilots.createIndex({"id":1})

We tune our performance to

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'AirlineDetails.pilots',
    indexFilterSet: false,
    parsedQuery: { '$where': Code("this.id") },
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'COLLSCAN',
      filter: { '$where': Code("this.id") },
      direction: 'forward'
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 7,
    executionTimeMillis: 1,
    totalKeysExamined: 0,
    totalDocsExamined: 7,
```

```
  executionStages: {
    stage: 'COLLSCAN',
    filter: { '$where': Code("this.id") },
    nReturned: 7,
    executionTimeMillisEstimate: 2,
    works: 9,
    advanced: 7,
    needTime: 1,
    needYield: 0,
    saveState: 0,
    restoreState: 0,
    isEOF: 1,
    direction: 'forward',
    docsExamined: 7
  }
},
command: {
  find: 'pilots',
  filter: { '$where': 'this.id' },
  '$db': 'AirlineDetails'
},
serverInfo: {
  host: 'LAPTOP-93ALV4L6',
  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
}
```


As we see in the highllighted area our executionTimeMillis has dropped from 44ms to 1ms

Profiling: Profiler output saved in group20_profiler_output.js as the file and the output contents are huge.