ECS650U/ECS789P - SEMI-STRUCTURED DATA AND ADVANCED DATA MODELLING – 2021/22

COURSEWORK 2 DATABASE DEVELOPMENT AND PERFORMANCE TUNING

CW2 -ECS789P_Group 23

Awais Jafar Iqbal Patekari

Zhengwu Ren

Mariia Ignashina

Uttkarsh Raj

ASSUMPTIONS

GENERAL ASSUMPTIONS:

Using timestamp for date/time is the industry standard. But since we do not have a front-end, we will use ISO just for the sake of readability.

We did not implement a complex id system in the DB prototype for the purpose of easier testing.

For more transparent testing, we only made a few flights and reservations all happening in a limited timeframe.

ASSUMPTIONS ABOUT PLANES:

The planes can either be working, or going under repair, or going under upgrade. Those states cannot overlap.

The date of the last upgrade of a plane will be updated only when the upgrade is over.

The length of service for a plane is calculated as the number of years that have passed since it was built.

Fields “year_of_commission” and “last_upgraded” are not required.

The range given for each plane is for a full fuel load.

ASSUMPTIONS ABOUT FLIGHTS:

Because of the weather, not all flights of the same distance will take the same time.

The pilot and the co-pilot must be different people picked from the pool of pilots.

The price of the flight ticket can be different depending on the proximity to the time of flight, demand, ticket agency, etc. It is external information. We will not include it in the flight collection to avoid confusion.

All flights are direct.

The distance between airports might vary depending on the route the plane takes.

The distance of the flight is in kilometers.

ASSUMPTIONS ABOUT AIRPORTS:

Because the terminals might change for flights a few times before said flights, we will not store that information in this, more general, database.

The charge of refuel is the same for every plane.

When counting the charge for a plane staying at an airport, the flight will be charged for the full hour if the plane stays at the airport for more than 30 mins.

All the airports are inside the UK (United Kingdom), so there is no currency problem for charges.

ASSUMPTIONS ABOUT BOOKINGS:

Not all users book tickets for themselves, so we must store the passengers separately.

All flights booked by the same user at the same time should pertain to the same passengers.

Because so far, we do not have cyber security, we will not store any personal information, such as nationality or documents of passengers.

Only an existing user can book tickets for a journey.

ASSUMPTIONS ABOUT USERS:

We only need user email to contact them in case of emergency, no other personal information.

ASSUMPTIONS ABOUT EMPLOYEES:

All employees hold no more than one position in the company.

The salary of staff is a fixed amount of money per month.

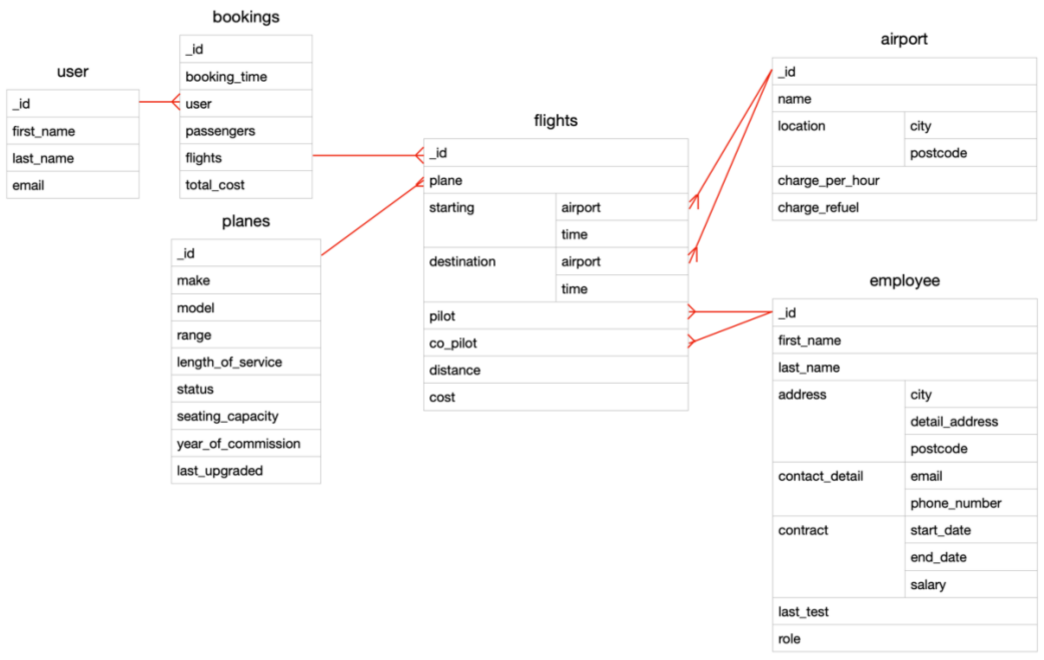
Fit for flight test will be valid for 3 years. Pilots can only be hired if they pass those tests regularly.

We assume that all pilots can drive all planes.

We assume that since the work is shift-based, the employees will mitigate their days-off and sick leave between themselves.

We assume that it will be the pilot’s job to make sure he is not overworked and always comes home.

ER DIAGRAM



COLLECTION SCHEMAS

PLANES:

```
{
    ″title″: ″planes″,
    ″properties″: {
        ″_id″: {
            ″bsonType″: ″Int″,
            ″required″: true
        },
        ″make″: {
            ″bsonType″: ″string″,
            ″required″: true
        },
        ″model″: {
            ″bsonType″: ″string″,
            ″required″: true
        },
        ″range″: {
            ″bsonType″: ″number″,
            ″required″: true
        },
        ″length_of_service″: {
            ″bsonType″: ″number″
            ″required″: true

        },

        ″status″: {

            ″bsonType″: ″string″

            ″required″: true

        },

        ″seating_capacity″: {

            ″bsonType″: ″number″

            ″required″: true

        },

        ″year_of_commission″: {

            ″bsonType″: ″timestamp″

            ″required″: false

        },

        ″last_upgraded″: {

            ″bsonType″: ″timestamp″

            ″required″: false

        }

    }

}
```

FLIGHTS

The plane field is an id of a plane, taken from the "planes" collection,

the starting and destination airport fields are ids of airports, taken from the "airport" collection, *the pilot and co-pilot are ids of pilots, taken from the “employee” collection:*

```
{
    ″title″: ″flights″,
    ″properties″: {
        ″_id″: {
            ″bsonType″: ″Int″
            ″required″: true
        },
        ″plane″: {
            ″bsonType″: ″Int″
            ″required″: true
        },
        ″starting″: {
            ″airport″: {
                ″bsonType″: ″Int″
                ″required″: true
            },
            ″time″: {
                ″bsonType″: ″timestamp″
                ″required″: true
            }
        },
        ″destination″: {
```

```
    ″airport″: {
        ″bsonType″: ″Int″
        ″required″: true
    },
    ″time″: {
        ″bsonType″: ″timestamp″
        ″required″: true
    }
},
″pilot″: {
    ″bsonType″: ″Int″
    ″required″: true
},
″co_pilot″: {
    ″bsonType″: ″Int″
    ″required″: true
},
″distance″: {
    ″bsonType″: ″number″
    ″required″: true
},
 ″cost″: {
    ″bsonType″: ″number″
```

```
                ″required″: true
            }
        }
      }
```

AIRPORTS:

```
    {
        ″title″: ″airport″,
        ″properties″: {
            ″_id″: {
                ″bsonType″: ″Int″,
                ″required″: true
            },
            ″name″: {
                ″bsonType″: ″string″,
                ″required″: true
            },
            ″location″: {
                 ″city″:{
                       ″bsonType″: ″string″,
                 ″required″: true
           },
    ″detail_address″:{
                 ″bsonType″: ″string″,
                    ″required″: false
              },
       ″postcode″:{
                    ″bsonType″: ″string″,
                    ″required″: true
              }
               },
               ″charge_per_hour″: {
                   ″bsonType″: ″number″
                   ″required″: true
               },
               ″charge_refuel″: {
                   ″bsonType″: ″number″
                   ″required″: true
               }
           }
         }
```

USERS:

```
       {
           ″title″: ″user″,
           ″properties″: {
               ″_id″: {
                "bsonType": "Int",
                "required": true
            },
            "first_name": {
                "bsonType": "string",
                "required": true
            },
            "last_name": {
                "bsonType": "string",
                "required": true
            },
            "email": {
                "bsonType": "string"
                "required": true
            }
        }
    }
```

BOOKINGS

The user field is an id of a user, taken from the "user" collection,

the flights fields are ids of flights, taken from the "flights" collection:

```
{
        "title": "bookings",
″properties″: {
    ″_id″: {
        ″bsonType″: ″Int″,
        ″required″: true
    },
    ″booking_time″: {
        ″bsonType″: ″timestamp″,
        ″required″: true
    },
    ″user″: {
        ″bsonType″: ″Int″,
        ″required″: true
    },
    ″passengers″: [
        {
            ″first_name″: {
                ″bsonType″: ″string″,
                ″required″: true
            },
            ″last_name″: {
                ″bsonType″: ″string″,
                ″required″: true
            }
```

```
                }
            ],
            ″flights″: [
                {
                    ″bsonType″: ″Int″,
                    ″required″: true
                }
            ],
            ″total_cost″: {
                ″bsonType″: ″number″
                ″required″: true
            }
        }
    }
```

EMPLOYEES:

```
{
        ″title″: ″employee″,
            ″properties″: {
                ″ _id″: {
                    ″bsonType″: ″Int″,
                    ″required″: true
                },
      ″first_name″: {
           ″bsonType″: ″string″,
           ″required″: true
       },
       ″last_name″: {
           ″bsonType″: ″string″,
           ″required″: true
       },
      ″address″: {
           ″city″:{
                ″bsonType″: ″string″,
                ″required″: true
  },
″detail_address″:{
                ″bsonType″: ″string″,
          ″required″: true
},
 ″postcode″:{
                ″bsonType″: ″string″,
                ″required″: true
  }
       },
      ″contact_detail″: {
        ″email ″: {
            ″bsonType″: ″string″,
            ″required″: false
        },
        ″phone_number″: {
            ″bsonType″: ″string″,
            ″required″: true
        },
    }
 ″contract ″: {
        ″start_date″: {
            ″bsonType″: ″timestamp″,
            ″required″: true
        },
        ″end_date″: {
            ″bsonType″: ″timestamp″,
            ″required″: true
        },
        ″salary ″: {
            ″bsonType″: ″number″,
            ″required″: true
        }
    },
                    ″last_test″: {
                        ″bsonType″: ″timestamp″,
                        ″required″: false
                    },
                    ″role″: {
                        ″bsonType″: ″String″,
                        ″required″: false
                    }
            }
        }
```

QUERIES

1) Find out the top 5 planes which have not been updated in the longest time? This is a basic query which will help us to know the planes which have not been upgraded for a long time.

```
=> db.planes.find().sort({ last_upgraded: 1 }).limit(2).pretty()
```

Expected output:

```
{
        "_id" : 7,
        "make" : "Antonov",
        "model" : "An-2",
        "range" : 1000,
        "length_of_service" : 34,
        "status" : "working",

        "seating_capacity" : 550,

        "year_of_commission" : ISODate("1987-12-11T00:00:00Z"),

        "last_upgraded" : ISODate("2000-06-05T00:00:00Z")

}

{

        "_id" : 10,

        "make" : "Antonov",

        "model" : "An-32",

        "range" : 1150,

        "length_of_service" : 28,

        "status" : "repaired",

        "seating_capacity" : 600,

        "year_of_commission" : ISODate("1993-02-09T00:00:00Z"),

        "last_upgraded" : ISODate("2001-07-23T00:00:00Z")

}
```

2) Find out the employees and their designation. This query will help us to generate the data of all the employees with their role in the airlines company.

```
=> db.employee.aggregate([{ $group: { _id: '$role' , count: {$sum: 1}}},
{$project: {_id: 0, role: '$_id',count:1}}])
```

Expected output:

```
{ count: 2, role: 'booking_clerk' }

{ count: 4, role: 'cabin_staff' }

{ count: 2, role: 'maintain_staff' }

{ count: 9, role: 'pilot'}
```

3) Find out where the most salary is spent in the airline. This query will help us to know the salary getting spent by the airlines for the separate roles.

```
=> db.employee.aggregate([{ $group: { _id: '$role', total_salary: {$sum:
'$contract.salary'}}},{$project:{_id: 0, role:'$_id',
total_salary:1}},{$sort:{total_salary: -1}}])
```

Expected output:

```
{ "total_salary" : 75900, "role" : "pilot" }

{ "total_salary" : 15300, "role" : "cabin_staff" }

{ "total_salary" : 8900, "role" : "maintain_staff" }

{ "total_salary" : 6500, "role" : "booking_clerk" }
```

4) Find out which user has spent how much amount while booking the tickets? This query will show the total amount spent by users while doing the bookings.

```
=> db.bookings.aggregate([{ $group: { _id: '$user', total_spent: {$sum:
'$total_cost'}}},{$sort: {total_spent: -
1}},{$lookup:{from:'user',localField:'_id',foreignField:'_id',as:'user'}},{$u
nwind:{path:'$user'}},{$project:{_id:0,'user.first_name':1,'user.last_name':1
,total_spent:1}}]).pretty()
```

Expected output:

```
{

        "total_spent" : 75000,

        "user" : {

                "first_name" : "Daniel",

                "last_name" : "Riccardo"

        }

}

{
        "total_spent" : 75000,
        "user" : {
                "first_name" : "Carlos",
                "last_name" : "Sainz"
        }
}
{
        "total_spent" : 40000,
        "user" : {
                "first_name" : "Lando",
                "last_name" : "Norris"
        }
}
{
        "total_spent" : 15000,
        "user" : {
                "first_name" : "Seb",
                "last_name" : "Vettle"
        }
}
{
        "total_spent" : 15000,
        "user" : {
                "first_name" : "Max",
                "last_name" : "Verstappen"
        }
}
```

5) Find out which airports are being used the most. This query will show us the data of the airports with their names and their usage.

```
=> db.airport.aggregate(
 [
    { $lookup:{
        from:"flights",
        let:{airport_id:"$_id"},
        pipeline:[
            {  $match:{
                $expr:
                  { $or:[
                      {$eq:["$starting.airport", "$$airport_id"]},
                      {$eq:["$destination.airport", "$$airport_id"]}
                  ]
                  }
            }
            },
            { $count: "airport_usage"}
        ],
        as: "fly_data"
    }},
    {$unwind:"$fly_data"},
    { $sort:{"fly_data.airport_usage":-1}},
    { $limit: 3},
    { $project: {_id:0,name:1,airport_usage:"$fly_data.airport_usage"}}
 ]
)
```

<u>Expected output:</u>

{ name: 'Manchester Airport', airport_usage: 9 }

{ name: 'Heathrow', airport_usage: 8 }

{ name: 'Gatwick', airport_usage: 8 }

6) Find out the top 3 pilots that have flown the most (distance is calculated here for both pilots and co-pilots). This query will show the data of the pilots with the calculated distance. It will show the first and last names of the pilots.

```
=> db.employee.aggregate(

[

    { $match:{role:"pilot"}},

    { $lookup:{

        from:"flights",

        let:{pilot_id:"$_id"},

        pipeline:[

            {  $match:{

                 $expr:

                   { $or:[

                       {$eq:["$pilot", "$$pilot_id"]},

                       {$eq:["$co_pilot", "$$pilot_id"]}

                   ]

                   }

            }

            },

            { $project: {distance:1}}

        ],

        as: "fly_data"

    }},
        {$unwind:"$fly_data"},

        { $group:{_id:"$_id",
first_name:{$addToSet:"$first_name"},last_name:{$addToSet:"$last_name"},total_dista
nce:{$sum:"$fly_data.distance"}}},

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

        { $limit: 3}

    ]

)
```

Expected output:

```
{ _id: 4, first_name: [ 'Jones' ],  last_name: [ 'Jackson' ],  total_distance: 4300 }

{ _id: 1, first_name: [ 'Smith' ], last_name: [ 'Johnson' ], total_distance: 4290 }

{ _id: 5, first_name: [ 'Garcia' ], last_name: [ 'Martin' ], total_distance: 3500 }
```

7) Find out the pilots/pilot whose 'fit-to-fly' duration is expired. This query will show the data of the pilots/pilot who needs to go for 'fit-to-fly' test again.

```
=> db.employee.aggregate({$match : {last_test: {$lt: new
Date(ISODate().getTime()- 1000*60*60*24*365*3)}}}).pretty()
```

Expected output:

```
{

        "_id" : 3,

        "first_name" : "Brown",

        "last_name" : "Moore",

        "address" : {

                "city" : "Manchester",

                "detail_address" : "Dakota",

                "postcode" : "MM1 2JL"

        },

        "contact_detail" : {
                "email" : "brown@foxmail.com",

                "phone_number" : "07543441245"

            },

            "contract" : {

                "start_date" : ISODate("2018-09-21T00:00:00Z"),

                "end_date" : ISODate("2023-09-20T00:00:00Z"),

                "salary" : 8500

            },

            "last_test" : ISODate("2018-07-30T00:00:00Z"),

            "role" : "pilot"

        }
```

8) Find out how much Smith Johnson would earn in a month if the salary were 2 pounds/km for pilot and 1.5 pounds/km for co-pilot.

```
=> db.employee.aggregate(

[{ $match:

{ first_name: "Smith", last_name: "Johnson" } },

{$facet:{

"total":[

        { $lookup: { from: "flights", localField: "_id", foreignField: "pilot"
,as:"pilots"}},

        { $unwind:"$pilots"},

        {
$project:{"pilots.pilot":1,"pilots.distance":1,date:{$dateFromString:{dateString:"$
pilots.destination.time"}}}},

        { $match:{"date":{$gte:new Date("2021-12-01"),$lt:new Date("2022-01-
01")}}},

        { $group:{_id:"$pilots.pilot",total:{$sum: "$pilots.distance"}}},

        { $project:{_id:0,total:{$multiply:["$total",2]}}},
        { $addFields:{total:"$total"}}],

    "co_total":[

        { $lookup: { from: "flights", localField: "_id", foreignField: "co_pilot"
,as:"co_pilots"}},

        { $unwind:"$co_pilots"},

        {
$project:{"co_pilots.co_pilot":1,"co_pilots.distance":1,date:{$dateFromString:{date
String:"$co_pilots.destination.time"}}}},

        { $match:{"date":{$gte:new Date("2021-12-01"),$lt:new Date("2022-01-
01")}}},

        { $group:{_id:"$co_pilots.co_pilot",co_total:{$sum:
"$co_pilots.distance"}}},

        { $project:{_id:0,total:{$multiply:["$co_total",1.5]}}}]

        }

        },

    {$unwind:"$total"},

    {$unwind:"$co_total"},

    {$project:{total:{$add:["$total.total","$co_total.total"]}}}

    ]

)
```

<u>Expected output:</u>

Smit Johnson would earn:

{total: 7585}

9) Find out the total revenue of the airline. (Assuming all the unused planes are staying in the airline's depot which is free. If the starting airport is different from the time of plane travelling to there. And we are neglecting the time before the first flight and the last flight)

=> `db.flights.aggregate(`

```
    [
        {
$project:{"plane":1,"starting.airport":1,"destination.airport":1,start_date:{$dateF
romString:{dateString:"$starting.time"}},end_date:{$dateFromString:{dateString:"$de
stination.time"}}}},

        { $match:{"end_date":{$gte:new Date("2021-12-01"),$lt:new Date("2022-01-
01")}}},

        { $project
:{plane:"$plane",start_airport:"$starting.airport",start_time:"$start_date",end_air
port:"$destination.airport",end_time:"$end_date"}},

        { $sort:{"plane":1,"start_time":1}},

        { $group:{_id:"$plane", all: {$push: "$$ROOT"}}},

        {$addFields: {allWithIndex: {$zip: {inputs: ["$all", {$range: [0, {$size:
"$all"}]}]}}}},

        {$project: {

          pairs: {

               $map: {

                    input: "$allWithIndex",

                    in : {

                        current: {$arrayElemAt: ["$$this", 0]},

                        prev: {

                            $arrayElemAt: [

                                "$all",

                                {$max: [0, {$subtract: [{$arrayElemAt: ["$$this",
1]}, 1]}]}

                            ]

                        }

                    }

               }

          }

        }},

        { $unwind: "$pairs"},
        {
$project:{_id:1,pairs:1,"end_time_mod":{$cond:{if:{$eq:["$pairs.current._id","$pair
s.prev._id"]},then:"$pairs.prev.start_time",else:"$pairs.prev.end_time"}}}},

        { $group: {

          _id: {plane: "$_id", aiport: "$pairs.current.start_airport"},

          stay_time: {$sum: {$subtract: ["$pairs.current.start_time",
"$end_time_mod"]}},

        }},

        { $group: {_id:"$_id.aiport",total_stay:{$sum:"$stay_time"}}},

        { $project: {_id:1,
total_stay:{$round:[{$divide:["$total_stay",3600000]},0]}}},

        {
$lookup:{from:"airport",localField:"_id",foreignField:"_id",as:"airport"}},

        { $unwind:"$airport"},

        {
$project:{airport_cost:{$sum:{$multiply:["$total_stay","$airport.charge_per_hour"]}
}}},

        { $group:{_id:null,total_airport_stay_cost:{$sum:"$airport_cost"}}},

        { $lookup:{

            from:"employee",

            pipeline:[

                {$group:{_id:null,total_salary:{$sum:"$contract.salary"}}},

                {$project:{total_salary:1,_id:0}}

            ],

            as:"employee_salary"}},

        { $unwind:"$employee_salary"},

        { $lookup:{

            from:"bookings",

            pipeline:[

                {$group:{_id:null,income:{$sum:"$total_cost"}}},
                {$project:{income:1,_id:0}}

            ],

            as:"total_income"}},

        { $unwind:"$total_income"},

        { $lookup:{

            from:"flights",

            pipeline:[

                {
$project:{"plane":1,"starting.airport":1,"destination.airport":1,start_date:{$dateF
romString:{dateString:"$starting.time"}},end_date:{$dateFromString:{dateString:"$de
stination.time"}}}},

                { $match:{"end_date":{$gte:new Date("2021-12-01"),$lt:new
Date("2022-01-01")}}},

                { $group:{_id:"$starting.airport",use_times:{$sum:1}}},

                {
$lookup:{from:"airport",localField:"_id",foreignField:"_id",as:"airport"}},

                { $unwind:"$airport"},

                {
$project:{airport_cost:{$sum:{$multiply:["$use_times","$airport.charge_refuel"]}}}}
,

                {
$group:{_id:null,total_airport_fuel_cost:{$sum:"$airport_cost"}}},

                {$project:{total_airport_fuel_cost:1,_id:0}}

            ],

            as:"total_fuel_cost"

        }},

        {$unwind:"$total_fuel_cost"},


{$project:{revenue:{$subtract:["$total_income.income",{$add:["$total_airport_stay_c
ost","$total_fuel_cost.total_airport_fuel_cost","$employee_salary.total_salary"]}]}
}}

    ]
```

)

Expected output:

Total revenue will be:

```
{ "_id" : null, "revenue" : -663860 }
```

10) Find out all the employees who are using Gmail? This query will show the email address of the employees who are using Gmail.

```
=> db.employee.find({'contact_detail.email':{$exists:
true},'contact_detail.email':/gmail/},{_id:0,first_name: 1, last_name: 1,
contact_detail: 1})
```

Expected output:

```
{ first_name: 'Smith',

  last_name: 'Johnson',

  contact_detail: { email: 'smith@gmail.com', phone_number: '07543440380' } }

{ first_name: 'Jones',

  last_name: 'Jackson',

  contact_detail: { email: 'jones@gmail.com', phone_number: '07543640463' } }

{ first_name: 'Garcia',

  last_name: 'Martin',

  contact_detail: { email: 'gracia@gmail.com', phone_number: '07543683904' } }

{ first_name: 'Lopez',

  last_name: 'Lewis',

  contact_detail: { email: 'lopez@gmail.com', phone_number: '07543156923' } }
```

11) Find out the total amount airline is getting from the bookings done by the users.

```
=> db.bookings.aggregate({ $group: {_id: null, sum: { $sum: "$total_cost"}}})
```

Expected output:

The airline is getting total sum of 220000

```
{ _id: null, sum: 220000}
```

12) Find out all the planes that hold more people than they fly the kilometers.

```
=> db.planes.find({            $where: "this.range && this.seating_capacity &&
this.range <= this.seating_capacity"
  })
```

Expected output:

All the planes are listed below:

```
{ _id: 1,
  make: 'Boeing',
  model: '747',
  range: 700,
  length_of_service: 22,
  status: 'working',
  seating_capacity: 800,
  year_of_commission: 1999-03-24T00:00:00.000Z,
  last_upgraded: 2017-03-02T00:00:00.000Z }
{ _id: 2,
  make: 'Boeing',
  model: '777',
  range: 900,
  length_of_service: 9,
  status: 'repaired',
  seating_capacity: 900,
  year_of_commission: 2012-07-14T00:00:00.000Z,
  last_upgraded: 2012-07-14T00:00:00.000Z }
{ _id: 3,
  make: 'Boeing',
  model: '747',
  range: 700,
  length_of_service: 24,
  status: 'working',
  seating_capacity: 800,
  year_of_commission: 1997-11-19T00:00:00.000Z,
  last_upgraded: 2010-01-30T00:00:00.000Z }
{ _id: 4,
  make: 'Airbus',
  model: 'A380',
  range: 600,
  length_of_service: 10,
  status: 'upgraded',
  seating_capacity: 700,
  year_of_commission: 2011-04-12T00:00:00.000Z,
  last_upgraded: 2011-04-12T00:00:00.000Z }
{ _id: 5,
  make: 'Airbus',
  model: 'ACJ319',
  range: 650,
  length_of_service: 17,
  status: 'working',
  seating_capacity: 650,
  year_of_commission: 2004-05-22T00:00:00.000Z,
```

last_upgraded: 2013-04-26T00:00:00.000Z }

13) Find out the flights that have not been booked.

```
=> db.flights.aggregate(
    [
        { $lookup:{
            from:"bookings",
            let:{flight_id:"$_id"},
            pipeline:[
                {$unwind:"$flights"},
                {$match:{$expr:{$eq:["$flights","$$flight_id"]}}},
                {$count:"booking_times"},
                {$project:{_id:0,booking_times:1}}
            ],
            as:"booking_data"
        }},
        {$match:{booking_data:[]}},
        {$project:{_id:1,plane:1,starting:1,destination:1}}
    ]
).pretty()
```

Expected output:

```
{
        "_id" : 3,
        "plane" : 14,
        "starting" : {
                "airport" : 4,
		"time" : "2021-12-17T10:00:00Z"
	},
	"destination" : {
		"airport" : 2,
		"time" : "2021-12-17T18:00:00Z"
	}
}
{
	"_id" : 4,
	"plane" : 7,
	"starting" : {
		"airport" : 5,
		"time" : "2021-12-12T11:15:00Z"
	},
	"destination" : {
		"airport" : 1,
		"time" : "2021-12-12T13:30:00Z"
	}
}
{
	"_id" : 5,
	"plane" : 1,
	"starting" : {
		"airport" : 5,
		"time" : "2021-12-17T11:45:00Z"
	},
	"destination" : {
		"airport" : 3,
                "time" : "2021-12-17T16:30:00Z"
        }
}
{
        "_id" : 7,
        "plane" : 11,
        "starting" : {
                "airport" : 6,
                "time" : "2021-12-12T17:15:00Z"
        },
        "destination" : {
                "airport" : 3,
                "time" : "2021-12-13T09:15:00Z"
        }
}
{
        "_id" : 8,
        "plane" : 5,
        "starting" : {
                "airport" : 4,
                "time" : "2021-12-10T13:00:00Z"
        },
        "destination" : {
                "airport" : 1,
                "time" : "2021-12-10T17:30:00Z"
        }
}
{
		"_id" : 9,
		"plane" : 13,
		"starting" : {
			"airport" : 2,
			"time" : "2021-12-16T21:30:00Z"
		},
		"destination" : {
			"airport" : 3,
			"time" : "2021-12-17T00:45:00Z"
		}
	}
	{
		"_id" : 10,
		"plane" : 14,
		"starting" : {
			"airport" : 2,
			"time" : "2021-12-18T09:00:00Z"
		},
		"destination" : {
			"airport" : 4,
			"time" : "2021-12-19T17:15:00Z"
		}
	}
	{
		"_id" : 11,
		"plane" : 7,
		"starting" : {
			"airport" : 1,
                "time" : "2021-12-19T16:30:00Z"
        },
        "destination" : {
                "airport" : 5,
                "time" : "2021-12-19T18:45:00Z"
        }
}
{
        "_id" : 12,
        "plane" : 14,
        "starting" : {
                "airport" : 1,
                "time" : "2021-12-10T06:15:00Z"
        },
        "destination" : {
                "airport" : 6,
                "time" : "2021-12-10T11:00:00Z"
        }
}
{
        "_id" : 13,
        "plane" : 14,
        "starting" : {
                "airport" : 6,
                "time" : "2021-12-10T13:00:00Z"
        },
        "destination" : {
                "airport" : 5,
                "time" : "2021-12-10T15:00:00Z"
        }
}
{
        "_id" : 16,
        "plane" : 13,
        "starting" : {
                "airport" : 1,
                "time" : "2021-12-14T23:45:00Z"
        },
        "destination" : {
                "airport" : 3,
                "time" : "2021-12-15T04:00:00Z"
        }
}
{
        "_id" : 17,
        "plane" : 13,
        "starting" : {
                "airport" : 5,
                "time" : "2021-12-12T21:15:00Z"
        },
        "destination" : {
                "airport" : 1,
                "time" : "2021-12-13T01:00:00Z"
        }
}
{
        "_id" : 18,
        "plane" : 13,
        "starting" : {
                "airport" : 3,
                "time" : "2021-12-15T07:30:00Z"
        },
        "destination" : {
                "airport" : 2,
                "time" : "2021-12-15T11:15:00Z"
        }
}
{
        "_id" : 19,
        "plane" : 8,
        "starting" : {
                "airport" : 4,
                "time" : "2021-12-12T10:00:00Z"
        },
        "destination" : {
                "airport" : 2,
                "time" : "2021-12-12T18:00:00Z"
        }
}
{
        "_id" : 20,
        "plane" : 6,
        "starting" : {
                "airport" : 3,
                "time" : "2021-12-09T13:00:00Z"
        },
        "destination" : {
                "airport" : 5,
                "time" : "2021-12-09T17:15:00Z"
        }
}
{
        "_id" : 21,
        "plane" : 9,
        "starting" : {
                "airport" : 6,
                "time" : "2021-12-14T12:15:00Z"
        },
        "destination" : {
                "airport" : 1,
                "time" : "2021-12-14T16:45:00Z"
        }
}
```

Note: We are showing a total of 13 queries.

EXPLAIN UTILITY AND INDEXES:

The aim of using explain() is to find out how to improve the query. In this section, we will attempt to determine if we're used the appropriate index and if we need to optimise other aspects of the queries or the data model. An explain plan allows us to understand fully the performance implications of the queries we have created. Looking at all the information returned by explain() we can see find out stuff like:

- how many documents were scanned
- how many documents were returned
- which index was used
- how long the query took to be executed
- which alternative execution plans were evaluated

The MongoDB database engine does its best to apply its own optimizations to the aggregate pipeline at run time. However, there may be some tweaks we can make ourselves. The database engine must not optimize the pipeline in such a way that it risks changing the behaviour and results of the pipeline. Database engines do not always have the additional context our brain has regarding current business problems. We may not be able to make certain decisions about which pipeline changes we need to apply to make it faster. We can fill this gap with an aggregate explain plan. This allows us to understand the database engine optimizations that are being applied and identify other potential optimizations that we can manually implement in our pipeline.

Below are the outputs from the explain() query. We have only added relevant fields from the profiler output to keep the report from becoming too long and verbose. The complete output of the explain() queries have been added to *explainerOutput.js* file for reference.

QUERY 6

```
"queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "coursework.employee",
        "indexFilterSet" : false,
        "parsedQuery" : {
                "role" : {
                        "$eq" : "pilot"
                }
        },
        "queryHash" : "D3D43117",
        "planCacheKey" : "D3D43117",
        "winningPlan" : {
                "stage" : "PROJECTION_DEFAULT",
                "transformBy" : {
                        "_id" : 1,
                        "first_name" : 1,
                        "fly_data.distance" : 1,
				"last_name" : 1
			},
			"inputStage" : {
				"stage" : "COLLSCAN",
				"filter" : {
					"role" : {
						"$eq" : "pilot"
					}
				},
				"direction" : "forward"
			}
		},
		"rejectedPlans" : [ ]
	},
	"executionStats" : {
		"executionSuccess" : true,
		"nReturned" : 9,
		"executionTimeMillis" : 4,
		"totalKeysExamined" : 0,
		"totalDocsExamined" : 17,
		"executionStages" : {
			"stage" : "PROJECTION_DEFAULT",
			"nReturned" : 9,
			"executionTimeMillisEstimate" : 0,
			"works" : 19,
			"advanced" : 9,
			"needTime" : 9,
			"needYield" : 0,
			"saveState" : 1,
			"restoreState" : 1,
			"isEOF" : 1,
			"transformBy" : {
				"_id" : 1,
				"first_name" : 1,
				"fly_data.distance" : 1,
				"last_name" : 1
			},
			"inputStage" : {
```

```
                                "stage" : "COLLSCAN",
                                "filter" : {
                                      "role" : {
                                            "$eq" : "pilot"
                                      }
                                },
                                "nReturned" : 9,
                                "executionTimeMillisEstimate" : 0,
                                "works" : 19,
                                "advanced" : 9,
                                "needTime" : 9,
                                "needYield" : 0,
                                "saveState" : 1,
                                "restoreState" : 1,
                                "isEOF" : 1,
                                "direction" : "forward",
                                "docsExamined" : 17
                          }
                    }
              }
```

The query is applied to the 'employee' collection where the winning plan is PROJECTION_DEFAULT that takes a COLLSCAN stage as input. The query uses COLLSCAN to filter through the documents in the collection to satisfy the $eq condition. The output of this (9 documents) is fed to PROJECTION_DEFAULT stage which returns the final output. The total execution time is 4 milliseconds.

EXPLAIN() AFTER ADDING INDEX

```
db.employee.createIndex({'role':1},{name:'roleIndex'})
```

```
              "queryPlanner" : {
                    "plannerVersion" : 1,
                    "namespace" : "coursework.employee",
                    "indexFilterSet" : false,
                    "parsedQuery" : {
                          "role" : {
                                "$eq" : "pilot"
                          }
```

```
        },
        "queryHash" : "D3D43117",
        "planCacheKey" : "EB1EED8D",
        "winningPlan" : {
              "stage" : "PROJECTION_DEFAULT",
              "transformBy" : {
                    "_id" : 1,
                    "first_name" : 1,
                    "fly_data.distance" : 1,
                    "last_name" : 1
              },
              "inputStage" : {
                    "stage" : "FETCH",
                    "inputStage" : {
                          "stage" : "IXSCAN",
                          "keyPattern" : {
                                "role" : 1
                          },
                          "indexName" : "roleIndex",
                          "isMultiKey" : false,
                          "multiKeyPaths" : {
                                "role" : [ ]
                          },
                          "isUnique" : false,
                          "isSparse" : false,
                          "isPartial" : false,
                          "indexVersion" : 2,
                          "direction" : "forward",
                          "indexBounds" : {
                                "role" : [
                                      "[\"pilot\",
\"pilot\"]"
                                ]
                          }
                    }
              }
        },
        "rejectedPlans" : [ ]
```

```
},
"executionStats" : {
	"executionSuccess" : true,
	"nReturned" : 9,
	"executionTimeMillis" : 3,
	"totalKeysExamined" : 9,
	"totalDocsExamined" : 9,
	"executionStages" : {
		"stage" : "PROJECTION_DEFAULT",
		"nReturned" : 9,
		"executionTimeMillisEstimate" : 0,
		"works" : 10,
		"advanced" : 9,
		"needTime" : 0,
		"needYield" : 0,
		"saveState" : 1,
		"restoreState" : 1,
		"isEOF" : 1,
		"transformBy" : {
			"_id" : 1,
			"first_name" : 1,
			"fly_data.distance" : 1,
			"last_name" : 1
		},
		"inputStage" : {
			"stage" : "FETCH",
			"nReturned" : 9,
			"executionTimeMillisEstimate" : 0,
			"works" : 10,
			"advanced" : 9,
			"needTime" : 0,
			"needYield" : 0,
			"saveState" : 1,
			"restoreState" : 1,
			"isEOF" : 1,
			"docsExamined" : 9,
			"alreadyHasObj" : 0,
			"inputStage" : {
                }

                                "stage" : "IXSCAN",
                                "nReturned" : 9,
                                "executionTimeMillisEstimate" : 0,
                                "works" : 10,
                                "advanced" : 9,
                                "needTime" : 0,
                                "needYield" : 0,
                                "saveState" : 1,
                                "restoreState" : 1,
                                "isEOF" : 1,
                                "keyPattern" : {
                                    "role" : 1
                                },
                                "indexName" : "roleIndex",
                                "isMultiKey" : false,
                                "multiKeyPaths" : {
                                    "role" : [ ]
                                },
                                "isUnique" : false,
                                "isSparse" : false,
                                "isPartial" : false,
                                "indexVersion" : 2,
                                "direction" : "forward",
                                "indexBounds" : {
                                    "role" : [
                                        "[\"pilot\",
\"pilot\"]"

                                    ]
                                },
                                "keysExamined" : 9,
                                "seeks" : 1,
                                "dupsTested" : 0,
                                "dupsDropped" : 0
                            }
                        }
                    }
                }
```

After adding an index, the winning plan is PROJECTION_DEFAULT. The query first scans the index keys using IXSCAN and then retrieves the relevant documents using FETCH. The output of this is sent to the PROJECTION_DEFAULT as before. The execution time is reduced to 3 milliseconds. This is because we are storing the 'role' field in the employee collection as an index as shown by the 'keyPattern' field of the explainer output. This reduces the time taken in fetching documents based on role. Although the reduction in execution time is not a lot, this will improve considerably when we add more employees to our collection. This is because when the number of documents is less, the performance of COLLSCAN is similar to IXSCAN.

QUERY 10

```
"queryPlanner" : {

            "plannerVersion" : 1,

            "namespace" : "coursework.employee",

            "indexFilterSet" : false,

            "parsedQuery" : {

                  "contact_detail.email" : {

                        "$regex" : "gmail"

                  }

            },

            "winningPlan" : {

                  "stage" : "PROJECTION_SIMPLE",

                  "transformBy" : {

                        "_id" : 0,

                        "first_name" : 1,

                        "last_name" : 1,

                        "contact_detail" : 1

                  },

                  "inputStage" : {

                        "stage" : "COLLSCAN",

                        "filter" : {

                              "contact_detail.email" : {

                                    "$regex" : "gmail"
```

```
					}
				},
				"direction" : "forward"
			}
		},
		"rejectedPlans" : [ ]
	},
	"executionStats" : {
		"executionSuccess" : true,
		"nReturned" : 4,
		"executionTimeMillis" : 0,
		"totalKeysExamined" : 0,
		"totalDocsExamined" : 17,
		"executionStages" : {
			"stage" : "PROJECTION_SIMPLE",
			"nReturned" : 4,
			"executionTimeMillisEstimate" : 0,
			"works" : 19,
			"advanced" : 4,
			"needTime" : 14,
			"needYield" : 0,
			"saveState" : 0,
			"restoreState" : 0,
			"isEOF" : 1,
			"transformBy" : {
				"_id" : 0,
				"first_name" : 1,
				"last_name" : 1,
                        "contact_detail" : 1
                },
                "inputStage" : {
                        "stage" : "COLLSCAN",
                        "filter" : {
                                "contact_detail.email" : {
                                        "$regex" : "gmail"
                                }
                        },
                        "nReturned" : 4,
                        "executionTimeMillisEstimate" : 0,
                        "works" : 19,
                        "advanced" : 4,
                        "needTime" : 14,
                        "needYield" : 0,
                        "saveState" : 0,
                        "restoreState" : 0,
                        "isEOF" : 1,
                        "direction" : "forward",
                        "docsExamined" : 17
                }
        }
```

The query is applied to the ‘employee’ collection where the winning plan is PROJECTION_DEFAULT that takes a COLLSCAN stage as input. The query uses COLLSCAN to filter through the documents in the collection to satisfy the $regex condition. The output of this (4 documents) is fed to PROJECTION_DEFAULT stage which returns the final output. The total execution time is 0 milliseconds as given by the ‘executionTimeMillis’ field.

EXPLAIN() AFTER ADDING INDEX

```
db.employee.createIndex({'contact_detail.email':1},{name: 'emailIndex'})
```

```
      "queryPlanner" : {
            "plannerVersion" : 1,
            "namespace" : "coursework.employee",
            "indexFilterSet" : false,
            "parsedQuery" : {
                  "contact_detail.email" : {
                        "$regex" : "gmail"
                  }
            },
            "winningPlan" : {
                  "stage" : "PROJECTION_SIMPLE",
                  "transformBy" : {
                        "_id" : 0,
                        "first_name" : 1,
                        "last_name" : 1,
                        "contact_detail" : 1
                  },
                  "inputStage" : {
                        "stage" : "FETCH",
                        "inputStage" : {
                              "stage" : "IXSCAN",
                              "filter" : {
                                    "contact_detail.email" : {
                                          "$regex" : "gmail"
                                    }
                              },
                        "keyPattern" : {
                              "contact_detail.email" : 1
                        },
                        "indexName" : "emailIndex",
                        "isMultiKey" : false,
                        "multiKeyPaths" : {
                              "contact_detail.email" : [ ]
                        },
                        "isUnique" : false,
                        "isSparse" : false,
                        "isPartial" : false,
                        "indexVersion" : 2,
                        "direction" : "forward",
                        "indexBounds" : {
                              "contact_detail.email" : [
                                    "[\"\", {})",
                                    "[/gmail/, /gmail/]"
                              ]
                        }
                  }
            }
      },
      "rejectedPlans" : [ ]
},
"executionStats" : {
      "executionSuccess" : true,
      "nReturned" : 4,
      "executionTimeMillis" : 0,
"totalKeysExamined" : 7,
"totalDocsExamined" : 4,
"executionStages" : {
      "stage" : "PROJECTION_SIMPLE",
      "nReturned" : 4,
      "executionTimeMillisEstimate" : 0,
      "works" : 8,
      "advanced" : 4,
      "needTime" : 3,
      "needYield" : 0,
      "saveState" : 0,
      "restoreState" : 0,
      "isEOF" : 1,
      "transformBy" : {
            "_id" : 0,
            "first_name" : 1,
            "last_name" : 1,
            "contact_detail" : 1
      },
      "inputStage" : {
            "stage" : "FETCH",
            "nReturned" : 4,
            "executionTimeMillisEstimate" : 0,
            "works" : 8,
            "advanced" : 4,
            "needTime" : 3,
            "needYield" : 0,
            "saveState" : 0,
"restoreState" : 0,
"isEOF" : 1,
"docsExamined" : 4,
"alreadyHasObj" : 0,
"inputStage" : {
      "stage" : "IXSCAN",
      "filter" : {
            "contact_detail.email" : {
                  "$regex" : "gmail"
            }
      },
      "nReturned" : 4,
      "executionTimeMillisEstimate" : 0,
      "works" : 8,
      "advanced" : 4,
      "needTime" : 3,
      "needYield" : 0,
      "saveState" : 0,
      "restoreState" : 0,
      "isEOF" : 1,
      "keyPattern" : {
            "contact_detail.email" : 1
      },
      "indexName" : "emailIndex",
      "isMultiKey" : false,
      "multiKeyPaths" : {
            "contact_detail.email" : [ ]
      },
                                "isUnique" : false,

                                "isSparse" : false,

                                "isPartial" : false,

                                "indexVersion" : 2,

                                "direction" : "forward",

                                "indexBounds" : {

                                      "contact_detail.email" : [

                                            "[\"\", {})",

                                            "[/gmail/, /gmail/]"

                                      ]

                                },

                                "keysExamined" : 7,

                                "seeks" : 1,

                                "dupsTested" : 0,

                                "dupsDropped" : 0

                          }

                    }

              }
```

After adding an index for the ‘contact_detail.email’ field, the winning plan is PROJECTION_DEFAULT. The query first scans the index keys using IXSCAN and then retrieves the relevant documents using FETCH. The output of this is sent to the PROJECTION_DEFAULT as before. The execution time stays 0 milliseconds. This is because we are storing the ‘contact_detail.email’ field in the employee collection as an index as shown by the ‘keyPattern’ field of the explainer output. This reduces the time taken in fetching documents based on email.

MONGODB PROFILER

MongoDB Profiler is a built-in tool that provides real-world query-level insights. This allows you to analyse all queries executed by the database system. Profilers typically store all data in the system.profile collection. This collection can be queried like any other regular MongoDB collection. The profiler has three levels of profiling. By default, the profiler level is set to 0 for any database.

- Level 0 - Profiler will not log any data
- Level 1 - Profiler will log only slow operations above some threshold
- Level 2 - Profiler will log all the operations

Let's set the profiling level to 2.

db.setProfilingLevel(2)

We will run a couple of queries and check the output of the profiler. We will only be adding some relevant fields in the report. We have added a file *profilerOutput.js* with the detailed profiler outputs –

- Query 8

```
{

        "op" : "command",

        "ns" : "coursework.employee",

        "command" : {


        },

        "keysExamined" : 0,

        "docsExamined" : 17,

        "cursorExhausted" : true,

        "numYield" : 0,

        "nreturned" : 1,

        "queryHash" : "9D795539",

        "planCacheKey" : "9D795539",

        "locks" : {


        },

        "flowControl" : {


        },
        "responseLength" : 131,

        "protocol" : "op_msg",

        "millis" : 2,

        "planSummary" : "COLLSCAN",

        "ts" : ISODate("2021-12-21T18:35:45.229Z"),

        "client" : "127.0.0.1",

        "appName" : "MongoDB Shell",

        "allUsers" : [ ],

        "user" : ""

}
```

- Query 3

```
{

        "op" : "command",

        "ns" : "coursework.employee",

        "command" : {

        },

        "keysExamined" : 0,

        "docsExamined" : 17,

        "cursorExhausted" : true,

        "numYield" : 0,

        "nreturned" : 4,

        "locks" : {

        },

        "flowControl" : {

        },

        "responseLength" : 315,
        "protocol" : "op_msg",

        "millis" : 1,

        "planSummary" : "COLLSCAN",

        "ts" : ISODate("2021-12-21T18:48:27.140Z"),

        "client" : "127.0.0.1",

        "appName" : "MongoDB Shell",

        "allUsers" : [ ],

        "user" : ""

}
```

The operations are a “command” type, and the target collections are the employee collection. Neither of the queries scan through any index keys. This is shown by the “keysExamined” field. The planSummary of all the queries is COLLSCAN. This can be a costly operation if the number of documents in the collection is very large. numYield is 0, this means that the operation yielded 0 times to allow other operations to complete. The simplicity of these queries did not require reading the index, so the key could not be looked up. Query no 3 took 1 millisecond to execute and Query no 8 took 2 milliseconds to complete.

We can see that Query no 8 is slower than Query no 3. So we can decide to only track queries that are slower than 2 milliseconds. We feel that any query taking less time than 2 milliseconds is fast enough and does not need tracking. We can do this by setting the profiler level to 1 and specifying the threshold of slow operations–

```
db.setProfilingLevel(1,{slowms:2})
```

The profiler will only track queries taking 2 milliseconds or longer to execute. You can check the output of the profiler by querying into the system.profile collection.

```
db.system.profile.find().pretty()
```