ECS79 Semi-structured Data and Advanced Data Modelling

CWK 2: Coursework 2 Database development and performance tuning

Group 6

Assumptions:

Each of the tables below lists the key assumptions of the entities taken in consideration for each collection.

GENERAL
all the data time fields are in ISO format.
Custom ids are used to facilitate testing of the correctness of queries. These ids are incremental integer that start from 1.
All the date collection fields (except for the connection time of the aircraft and the last "fit to fly" date of the pilot) are within the range of 2021-11-1 to 2021-11-7. The scope generated by this limited scope is intended to be a real realization of the provided system.
The "suitable for flight" test is valid for half a year to distinguish whether the pilot is suitable for the flight co-pilot
The time for all queries is 2021-11-08T00:00Z, for the convenience of calculation.
Assume that all aircrafts at 2021-11-08T00:00Z will eventually need

to return to the company's headquarters-Gatwick Airport for inspection.

Employee
The company has reservation staff, maintenance staff, pilots and flight attendants responsible for each.
All employees' salaries are fixed values, and are weekly salary, in pounds sterling.
The pilot information consists of an embedded document that records the details of the pilot, including their address, other contact information and their employment records in the company, as well as the date of their last "fit to fly" test. The address is composed of embedded documents, which record the pilot' s address information; other contact information is composed of embedded documents, which record the pilot' s contact information; the company' s employment record is composed of embedded documents, which record the employee' s entry time and other information ; The date of the last "fit to fly" test records the date when the pilot passed the test for the last time, and the validity period of the "fit to fly" test is half a year.

Plane
The service date records the time when the aircraft began service. This can be used to determine the usage time of the aircraft.
flying_range records the flying distance of the aircraft at full load. The unit is kilometers.
status records the status of the aircraft, which may be work, maintenance or upgrade.
The number of seats in the same type of aircraft is the same.

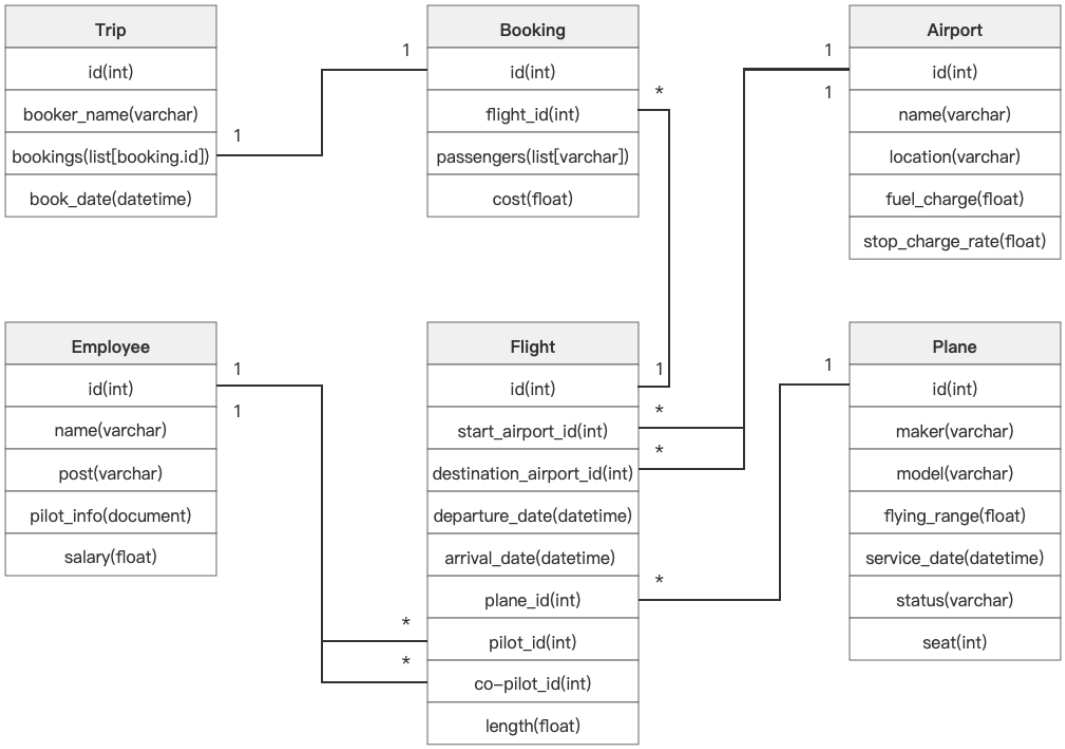
Airport
The location records the city where the airport is located.
The fuel charge records the cost of one refueling of the aircraft. The fee is fixed and may vary from airport to airport. Units are in British pounds.
The stop_charge rate records the hourly fee when the plane is parked at the airport. It may be different for different airports. The unit is pounds per hour.

Flight
length records the flight distance of the flight, in kilometers
The co-pilot must be a pilot who is within the validity period of the "fit to fly" test, and the main pilot does not need it.

Booking
One reservation record only corresponds to one flight. Orders involving multiple flights are represented by a list of multiple reservation records.
A reservation record can include multiple passengers
The price of the same flight can be different, and the unit is GBP.

Trip
A journey consists of a set of reservations.
A customer can book multiple journeys.

Diagram：



Collections：

Each employee(especially pilots) has 1-to-many relationship with flight because very flight will be operated by many piolets.

Employee

-id int

-name varchar

-post varchar

-pilot_info: document

-address: document

-postcode

-street

-city

-contact_details: document

-email

-phone

-mobile

-employment_records[]

-start_date

-end_date

-lastTestDate

-salary float

Each airport has 1-to-many relationship with flight, due to the every airport can be some flights' start airport and other flights' destination airport.

Airport

-id int

-name varchar

-location varchar

-fuel_charge float

-stop_charge_rate float

Plan

-id int

-maker varchar

-model varchar

-flying_range float

-service_date datetime

-status varchar

-seat int

Each flight has 1-to-many relationship with booking, because every flights will be booked many times.

Flight

-id int

-start_airlport_id int (references Airport.id)

-destination_airport_id. int (references Airport.id)

-departure_date datetime

-arrival_date datetime

-plane_id int (references Plan.id)

-pilot_id int (references Employee.id)

-co_pilot_id int (references Employee,id)

-length float

Booking

-id int

-flight_id int (references Flight.id)

-passengers[ ] list[varchar]

-cost float

Trip

-id int

-booker_name varchar

-bookings[ ] list[Booking.id](references Booking.id)

-book_date datetime

Queries:

1. Find how much the charge for stopping at an airport.

```
> db.Airport.aggregate([{
      $lookup: {
         from: "Flight",
         localField: "id",
         foreignField: "start_airport_id",
         as: "flight_from"
      }
   },
   {
      $lookup: {
         from: "Flight",
         localField: "id",
         foreignField: "destination_airport_id",
         as: "flight_to"
      }
   },
   {
      $project: {
         _id: 0,
         stop_charge: "1",
         charge1: {
            $reduce: {
               input: {
                  $map: {
                     input: "$flight_from",
                     as: "ff",
                     in: {
                        $multiply: [{
                              $divide: [{
                                    $subtract: [{
                                       $toDate:
"$$ff.departure_date"
                                    }, {
                                       $toDate: "2021-11-
01T"
                                    }]
                                 },
                                 3600000
                              ]
```

```
                                },
                                "$stop_charge_rate"
                            ]
                        }
                    }
                },
                initialValue: 0,
                in: {
                    $add: ["$$value", "$$this"]
                }
            }
        },
        charge2: {
            $reduce: {
                input: {
                    $map: {
                        input: "$flight_to",
                        as: "ft",
                        in: {
                            $multiply: [{
                                    $divide: [{
                                            $subtract: [{
                                                $toDate:
"$$ft.arrival_date"
                                            }, {
                                                $toDate: "2021-11-
01T"
                                            }]
                                        },
                                        3600000
                                    ]
                                },
                                "$stop_charge_rate"
                            ]
                        }
                    }
                },
                initialValue: 0,
                in: {
                    $add: ["$$value", "$$this"]
                }
            }
        }
    }
```

```
    },
    {
        $group: {
            _id: "$stop_charge",
            sum_charge1: {
                $sum: "$charge1"
            },
            sum_charge2: {
                $sum: "$charge2"
            }
        }
    }
])
```

Expect output: `{ "_id" : "1", "sum_charge1" : 5743.233333333334, "sum_charge2" : 7985.691666666667 }`

Tips: the final charge of stop is the 24 hours multiply the stop fee of airport2(due to all plane will return airport 2 finally), and then minus charge 2 and plus charge 1.

2. Find how much every trip will cost. This query provides obviously cost information about trip for the people who want to book.

```
> db.Trip.aggregate({
    $unwind: "$bookings"
}, {
    $lookup: {
        from: "Booking",
        localField: "bookings",
        foreignField: "id",
        as: "used"
    }
}, {
    $unwind: "$used"
}, {
    $group: {
        _id: "$id",
        moneySpent: {
            $sum: "$used.cost"
```

```
        }
    }
})
```

Expected output:

```
{ "_id" : 3, "moneySpent" : 1525 }
{ "_id" : 1, "moneySpent" : 1195 }
{ "_id" : 4, "moneySpent" : 2050 }
{ "_id" : 2, "moneySpent" : 1395 }
{ "_id" : 5, "moneySpent" : 1115 }
```

3. Find address and contact information of pilot whose id is 1. This query can be used to query pilot information quickly by change the id.

```
> db.Flight.aggregate([{
        $match: {
            id: 1
        }
    },
    {
        $lookup: {
            from: "Employee",
            localField: "pilot_id",
            foreignField: "id",
            as: "pilot"
        }
    },
    {
        $unwind: "$pilot"
    },
    {
        $project: {
            _id: 0,
            id: 1,
            pilot_id: 1,
            pilot_name: "$pilot.name",
            pilot_address: "$pilot.pilot_info.address",
            pilot_contact: "$pilot.pilot_info.contact_details"
```

```
      }
   }
])
```

Expected output: `{ "id" : 1, "pilot_id" : 1, "pilot_name" : "Puff Kineton", "pilot_address" : { "postcode" : "E3 4LL", "street" : "63 Pleasure Road", "city" : "London" }, "pilot_contact" : { "email" : "pkineton0@hostgator.com", "phone" : "4268527358", "mobile" : "+44 07653039504" } }`

4. Find the piolets who are eligible to be co-pilots who are in services and passed the test. This query is useful when the airplane company decide who can be the flight' s pilot.

```
> db.Employee.aggregate([{
      $match: {
         post: "pilot"
      }
   },
   {
      $project: {
         _id: 0,
         id: 1,
         name: 1,
         post: 1,
         end_date: "$pilot_info.employment_records.end_date",
         lastTestDate: "$pilot_info.lastTestDate"
      },
   },
   {
      $unwind: "$end_date"
   },
   {
      $match: {
         end_date: {
            "$gt": "2021-11-08T"
         },
         //Exclude departing pilots
```

```
         lastTestDate: {
            "$gt": "2021-05-09T"
         }
      } //The gap between 2021-05-09 and 2021-11-08 is 183
days, so pilots who passed last "fit for flying" test after
2021-05-09 are qualified.
   }
])
```

Expected output: `{ "id" : 1, "name" : "Puff Kineton", "post" :`

```
"pilot", "end_date" : "2026-07-01T", "lastTestDate" : "2021-
06-05T" }
{ "id" : 2, "name" : "Brigit Grover", "post" : "pilot",
"end_date" : "2023-07-01T", "lastTestDate" : "2021-08-24T" }
{ "id" : 5, "name" : "Margie Reidshaw", "post" : "pilot",
"end_date" : "2023-02-28T", "lastTestDate" : "2021-08-12T" }
{ "id" : 6, "name" : "Charmane Duffy", "post" : "pilot",
"end_date" : "2024-10-30T", "lastTestDate" : "2021-07-06T" }
{ "id" : 7, "name" : "Cybill Lorrie", "post" : "pilot",
"end_date" : "2025-07-31T", "lastTestDate" : "2021-09-22T" }
   { "id" : 8, "name" : "Mason Mount", "post" : "pilot",
   "end_date" : "2025-02-01T", "lastTestDate" : "2021-08-29T" }
```

5. Find what planes cannot be used.

```
> db.Plane.aggregate([{
   $match: {
      status: "working"
   }
}])
```

Expected output: `{ "_id" : ObjectId("61c359bd95e4db7a55bbe452"),`

```
"id" : 1, "maker" : "Boeing", "model" : "Boeing 747",
"flying_range" : 11250, "service_date" : "2003-03-20T",
"status" : "working", "seat" : 100 }
{ "_id" : ObjectId("61c359bd95e4db7a55bbe453"), "id" : 2,
"maker" : "Airbus", "model" : "Airbus A380", "flying_range" :
```

```
10500, "service_date" : "2005-10-22T", "status" : "working",
"seat" : 80 }
{ "_id" : ObjectId("61c359bd95e4db7a55bbe454"), "id" : 3,
"maker" : "Boeing", "model" : "Boeing 747", "flying_range" :
11250, "service_date" : "2012-09-26T", "status" : "working",
"seat" : 100 }
   { "_id" : ObjectId("61c359bd95e4db7a55bbe455"), "id" : 4,
   "maker" : "Boeing", "model" : "Boeing 737", "flying_range" :
   10250, "service_date" : "2015-01-02T", "status" : "working",
   "seat" : 120 }
```

6. Find the plane with the shortest services time.

```
> db.Plane.find({}).sort({
   service_date: -1
}).limit(1)
```

Expected outcome: `{ "_id" : ObjectId("61c359bd95e4db7a55bbe455"),`

```
"id" : 4, "maker" : "Boeing", "model" : "Boeing 737",
"flying_range" : 10250, "service_date" : "2015-01-02T",
"status" : "working", "seat" : 120 }
```

7. **Find the flights what the distance is over 5000 KM.**

```
> db.Flight.aggregate([{
   $match: {
      length: {
         $gt: 5000
      }
   }
}])
```

Expected outcome: `{ "_id" :`

```
ObjectId("61c359d595e4db7a55bbe458"), "id" : 1,
"start_airport_id" : 1, "destination_airport_id" : 3,
"departure_date" : "2021-11-01T07:11Z", "arrival_date" :
```

"2021-11-01T15:42Z", "plane_id" : 1, "pilot_id" : 1, "co-pilot_id" : 2, "length" : 5081 }
{ "_id" : ObjectId("61c359d595e4db7a55bbe45a"), "id" : 3, "start_airport_id" : 3, "destination_airport_id" : 5, "departure_date" : "2021-11-01T19:08Z", "arrival_date" : "2021-11-02T03:28Z", "plane_id" : 1, "pilot_id" : 2, "co-pilot_id" : 6, "length" : 5152.9 }
{ "_id" : ObjectId("61c359d595e4db7a55bbe45c"), "id" : 5, "start_airport_id" : 4, "destination_airport_id" : 1, "departure_date" : "2021-11-02T06:02Z", "arrival_date" : "2021-11-02T18:27Z", "plane_id" : 4, "pilot_id" : 7, "co-pilot_id" : 8, "length" : 7996.3 }
{ "_id" : ObjectId("61c359d595e4db7a55bbe45d"), "id" : 6, "start_airport_id" : 1, "destination_airport_id" : 4, "departure_date" : "2021-11-03T05:47Z", "arrival_date" : "2021-11-03T13:31Z", "plane_id" : 4, "pilot_id" : 8, "co-pilot_id" : 5, "length" : 7995.7 }
{ "_id" : ObjectId("61c359d595e4db7a55bbe45f"), "id" : 8, "start_airport_id" : 5, "destination_airport_id" : 3, "departure_date" : "2021-11-04T13:45Z", "arrival_date" : "2021-11-04T22:17Z", "plane_id" : 1, "pilot_id" : 6, "co-pilot_id" : 2, "length" : 5153 }
{ "_id" : ObjectId("61c359d595e4db7a55bbe461"), "id" : 10, "start_airport_id" : 3, "destination_airport_id" : 1, "departure_date" : "2021-11-05T08:36Z", "arrival_date" : "2021-11-05T16:50Z", "plane_id" : 1, "pilot_id" : 1, "co-pilot_id" : 6, "length" : 5080.6 }
{ "_id" : ObjectId("61c359d595e4db7a55bbe462"), "id" : 11, "start_airport_id" : 3, "destination_airport_id" : 5, "departure_date" : "2021-11-05T15:45Z", "arrival_date" : "2021-11-05T23:52Z", "plane_id" : 3, "pilot_id" : 7, "co-pilot_id" : 2, "length" : 5153.4 }
{ "_id" : ObjectId("61c359d595e4db7a55bbe463"), "id" : 12, "start_airport_id" : 5, "destination_airport_id" : 3, "departure_date" : "2021-11-06T12:09Z", "arrival_date" : "2021-11-06T20:20Z", "plane_id" : 2, "pilot_id" : 5, "co-pilot_id" : 7, "length" : 5153.2 }
{ "_id" : ObjectId("61c359d595e4db7a55bbe464"), "id" : 13, "start_airport_id" : 5, "destination_airport_id" : 2, "departure_date" : "2021-11-07T08:30Z", "arrival_date" : "2021-11-07T23:17Z", "plane_id" : 3, "pilot_id" : 2, "co-pilot_id" : 8, "length" : 9043.9 }
{ "_id" : ObjectId("61c359d595e4db7a55bbe467"), "id" : 16, "start_airport_id" : 4, "destination_airport_id" : 2,

```
"departure_date" : "2021-11-07T05:00Z", "arrival_date" :
"2021-11-07T18:09Z", "plane_id" : 4, "pilot_id" : 1, "co-
pilot_id" : 6, "length" : 8006.4 }
```

8. **Find the flight hours of pilots whose id is 1. This query provides a method to sort the flight hours of pilots by changing id.**

```
> db.Employee.aggregate([{
      $match: {
         id: 1
      }
   },
   {
      $lookup: {
         from: "Flight",
         localField: "id",
         foreignField: "pilot_id",
         as: "pilot"
      }
   },
   {
      $lookup: {
         from: "Flight",
         localField: "id",
         foreignField: "co-pilot_id",
         as: "co-pilot"
      }
   },
   {
      $project: {
         _id: 0,
         name: 1,
         hours_as_pilot: {
            $reduce: {
               input: {
                  $map: {
                     input: "$pilot",
                     as: "p",
                     in: {
                        $divide: [{
                              $subtract: [{
```

```
                                    $toDate: "$$p.arrival_date"
                                 }, {
                                    $toDate:
"$$p.departure_date"
                                 }]
                              },
                              3600000
                           ]
                        }
                     }
                  },
                  initialValue: 0,
                  in: {
                     $add: ["$$value", "$$this"]
                  }
               }
            },
            hours_as_copilot: {
               $reduce: {
                  input: {
                     $map: {
                        input: "$co-pilot",
                        as: "cp",
                        in: {
                           $divide: [{
                                 $subtract: [{
                                    $toDate:
"$$cp.arrival_date"
                                 }, {
                                    $toDate:
"$$cp.departure_date"
                                 }]
                              },
                              3600000
                           ]
                        }
                     }
                  },
                  initialValue: 0,
                  in: {
                     $add: ["$$value", "$$this"]
                  }
               }
            }
```

```
      }
   },
])
```

Expected output: `{ "name" : "Puff Kineton", "hours_as_pilot" : 29.9, "hours_as_copilot" : 10.933333333333334 }`

9. **Find the flight depart form the airport which id is 1 in 2021-11-01.**

```
> db.Flight.aggregate([{
   $match: {
      start_airport_id: 1,
      departure_date: {
         $gte: "2021-11-01T00:00Z",
         $lt: "2021-11-02T00:00Z"
      }
   },
}])
```

Expected outcome: `{ "_id" : ObjectId("61c359d595e4db7a55bbe458"), "id" : 1, "start_airport_id" : 1, "destination_airport_id" : 3, "departure_date" : "2021-11-01T07:11Z", "arrival_date" : "2021-11-01T15:42Z", "plane_id" : 1, "pilot_id" : 1, "co-pilot_id" : 2, "length" : 5081 }`

10. **Find the list of planes in descending order of maximum flight range.**

```
> db.Plane.find().sort({
```

```
   flying_range: -1
})
```

Expected outcome: `{ "_id" :`

```
ObjectId("61c359bd95e4db7a55bbe452"), "id" : 1, "maker" :
"Boeing", "model" : "Boeing 747", "flying_range" : 11250,
"service_date" : "2003-03-20T", "status" : "working", "seat" :
100 }
{ "_id" : ObjectId("61c359bd95e4db7a55bbe454"), "id" : 3,
"maker" : "Boeing", "model" : "Boeing 747", "flying_range" :
11250, "service_date" : "2012-09-26T", "status" : "working",
"seat" : 100 }
{ "_id" : ObjectId("61c359bd95e4db7a55bbe453"), "id" : 2,
"maker" : "Airbus", "model" : "Airbus A380", "flying_range" :
10500, "service_date" : "2005-10-22T", "status" : "working",
"seat" : 80 }
{ "_id" : ObjectId("61c359bd95e4db7a55bbe456"), "id" : 5,
"maker" : "Airbus", "model" : "Airbus A380", "flying_range" :
10500, "service_date" : "2001-02-14T", "status" : "repaired",
"seat" : 80 }
{ "_id" : ObjectId("61c359bd95e4db7a55bbe455"), "id" : 4,
"maker" : "Boeing", "model" : "Boeing 737", "flying_range" :
10250, "service_date" : "2015-01-02T", "status" : "working",
"seat" : 120 }
   { "_id" : ObjectId("61c359bd95e4db7a55bbe457"), "id" : 6,
   "maker" : "Airbus", "model" : "Airbus A350", "flying_range" :
   9800, "service_date" : "1999-06-17T", "status" : "upgraded",
   "seat" : 60 }
```

11. Find the model of plane with the least number of seats.

```
> db.Plane.find({}, {
   _id: 0,
   model: 1,
   seat: 1
}).sort({
   seat: 1
}).limit(1)
```

Expected outcome: `{ "model" : "Airbus A350", "seat" : 60 }`

12. Find the expenses of Paul Pogba.

```
> db.Trip.aggregate([{
      $match: {
         booker_name: "Paul Pogba"
      }
   },
   {
      $lookup: {
         from: "Booking",
         localField: "bookings",
         foreignField: "id",
         as: "totalcost"
      }
   },
   {
      $project: {
         _id: 0,
         booker_name: 1,
         totalcost: {
            $sum: "$totalcost.cost"
         },
      }
   }
])
```

Expected outcome: `{ "booker_name" : "Paul Pogba", "totalcost" : 1525 }`

Explain and profiling utilities:

Explain:

In this section we will using the explain utility to analyze the execution of 2 complicated queries to better understand the internal execution of each by the MongoDB system. No matter these queries with or without index, we will comment them. For reading easily, we will put the result text what worth to analyze and cut the query code text. But you can find the whole explainer result in explainer.json.

Query number 1 explainer about:


```
{
    "stages" : [
        {
            "$cursor" : {
                "queryPlanner" : {
                    "plannerVersion" : 1,
                    "namespace" : "test.Airport",
                    "indexFilterSet" : false,
                    "parsedQuery" : {

                    },
                    "queryHash" : "55302C7B",
                    "planCacheKey" : "55302C7B",
                    "winningPlan" : {
                        "stage" : "PROJECTION_SIMPLE",
                        "transformBy" : {
                            "flight_from" : 1,
                            "flight_to" : 1,
                            "id" : 1,
                            "stop_charge_rate" : 1,
                            "_id" : 0
                        },
                        "inputStage" : {
                            "stage" : "COLLSCAN",
                            "direction" : "forward"
                        }
                    },
                    "rejectedPlans" : [ ]
```

```
                }
            }
        },

    "serverInfo" : {
        "host" : "chenjingshudeMacBook-Air.local",
        "port" : 27017,
        "version" : "4.4.10",
        "gitVersion" :
"58971da1ef93435a9f62bf4708a81713def6e88c"
    },
    "ok" : 1
}
```

This query is applied to Airport collection, obviously this query did not use the index. Because the status of stage is COLLSCAN, this query will scan all the collection to sort. And the scan direction is forward. And no rejectionPlans.

Query number 8 explainer about:

```
{
    "stages" : [
        {
            "$cursor" : {
                "queryPlanner" : {
                    "plannerVersion" : 1,
                    "namespace" : "test.Employee",
                    "indexFilterSet" : false,
                    "parsedQuery" : {
                        "id" : {
                            "$eq" : 1
                        }
                    },
                    "queryHash" : "861B2C5E",
                    "planCacheKey" : "861B2C5E",
                    "winningPlan" : {
```

```
                        "stage" : "PROJECTION_SIMPLE",
                        "transformBy" : {
                            "co-pilot" : 1,
                            "id" : 1,
                            "name" : 1,
                            "pilot" : 1,
                            "_id" : 0
                        },
                        "inputStage" : {
                            "stage" : "COLLSCAN",
                            "filter" : {
                                "id" : {
                                    "$eq" : 1
                                }
                            },
                            "direction" : "forward"
                        }
                    },
                    "rejectedPlans" : [ ]
                }
            }
        },

    "serverInfo" : {
        "host" : "chenjingshudeMacBook-Air.local",
        "port" : 27017,
        "version" : "4.4.10",
        "gitVersion" :
"58971da1ef93435a9f62bf4708a81713def6e88c"
    },
    "ok" : 1
}
```


This query is applied in Employee collection. It did not use the index. Because the status of stage is COLLSCAN, this query will scan all the collection to sort. Meanwhile, the filter condition is id=1. The direction of scanning is forward.

Profiler:

In this section, the MongoDB profiler utility will be used to gather and analyses the details of the queries' operations and their performance. These queries are the same as last section. However, we will put in the most information of profiler output for reading easily. the whole profiler out put can be found in query_profile_output.json.

Query number 1 profiler output:


```
{
   "op" : "command",
   "ns" : "fly.Airport",
   "command" : {
      "aggregate" : "Airport",
      "pipeline" : [
         {
            "$lookup" : {
               "from" : "Flight",
               "localField" : "id",
               "foreignField" : "start_airport_id",
               "as" : "flight_from"
            }
         },
   "keysExamined" : NumberInt(0),
   "docsExamined" : NumberInt(165),
   "cursorExhausted" : true,
   "numYield" : NumberInt(0),
   "nreturned" : NumberInt(1),
   "queryHash" : "06E4134B",
   "planCacheKey" : "B1A95D42",
   "locks" : {
      "Global" : {
         "acquireCount" : {
            "r" : NumberLong(22)
         }
      },
      "Mutex" : {
         "acquireCount" : {
            "r" : NumberLong(22)
```

```
            }
        }
    },
    "flowControl" : {

    },
    "responseLength" : NumberInt(161),
    "protocol" : "op_msg",
    "millis" : NumberInt(1),
    "planSummary" : "COLLSCAN",
    "ts" : ISODate("2021-12-22T04:18:07.488+0000"),
    "client" : "127.0.0.1",
    "appName" : "Studio 3T",
    "allUsers" : [
        {
            "user" : "boss",
            "db" : "admin"
        }
    ],
    "user" : "boss@admin"
}
```

this query is aggregate operation which is applied in Airport collection. The return profile data shows that, because this query need to scan the whole collection, it took 1 milliseconds to execute. The numYield is 0, meaning that the operation was not interrupted by any read to the database. No keys were examined as the simplicity of this query didn't require any index read.

Query number 8 profiler output:

```
{
    "op" : "command",
    "ns" : "fly.Employee",
    "command" : {
        "aggregate" : "Employee",
```

```
        "pipeline" : [
            {
                "$match" : {
                    "id" : 1.0
                }
            },
    "keysExamined" : NumberInt(0),
    "docsExamined" : NumberInt(45),
    "cursorExhausted" : true,
    "numYield" : NumberInt(0),
    "nreturned" : NumberInt(1),
    "queryHash" : "771F3F74",
    "planCacheKey" : "ED61201A",
    "locks" : {
        "Global" : {
            "acquireCount" : {
                "r" : NumberLong(6)
            }
        },
        "Mutex" : {
            "acquireCount" : {
                "r" : NumberLong(6)
            }
        }
    },
    "flowControl" : {

    },
    "responseLength" : NumberInt(182),
    "protocol" : "op_msg",
    "millis" : NumberInt(0),
    "planSummary" : "COLLSCAN",
    "ts" : ISODate("2021-12-22T04:18:09.721+0000"),
    "client" : "127.0.0.1",
    "appName" : "Studio 3T",
    "allUsers" : [
        {
            "user" : "boss",
            "db" : "admin"
        }
    ],
    "user" : "boss@admin"
}
```

this query is aggregate operation which is applied in Employee collection. The return profile data shows that, although this query need to scan the whole collection, it took 0 milliseconds to execute. The numYield is 0, meaning that the operation was not interrupted by any read to the database. No keys were examined as the simplicity of this query didn’ t require any index read.