Queen Mary University of London

ECS789P Semi-Structured Data and Advance Data Modelling

Coursework 2: Database Modelling and Performance Tunning

Group 22: Temesgen Teklebrhan, David Stumbra, Remi Bahar, Tanishq Verma

Part One

A.Assumptions:

Throughout the coursework several assumptions were made in modelling and designing the schema of the airline database as well as on the value and type of data to be stored. The following tables give a brief list of the main assumptions made.

Bookings
Times given in UTC 0
Each booking is associated with a flight with a foreign identifier key.
Each passenger has a unique passport number in addition to booking ID.
A passenger may book a flight that is more than one stop, hence journeys are stored as array of flight IDs.
Total cost includes all the costs of the booking (flights, baggage, upgrades, etc) for all the journeys a passenger takes.

Employees
Employees are paid their standard monthly salary (salary/12).
Only the most recent job title of an employee is stored in employment_record.

The health_test refers to the date an employee passes their health and test and is therefore fit to fly.

Flights
A plane must refuel prior to departure.
Upon arriving at an airport, a plane only refuels if it takes another journey.
Planes are stored for free when not in use, so the airline company only pays for the duration planes are stopped over at an airport.
Every plane has a stopover duration of 1 hour prior to departure.

Airports
The cost to refuel depends only on the airport and not the particular plane (which may have varying fuel tank sizes) being refuelled.

B.MongoDB Schema

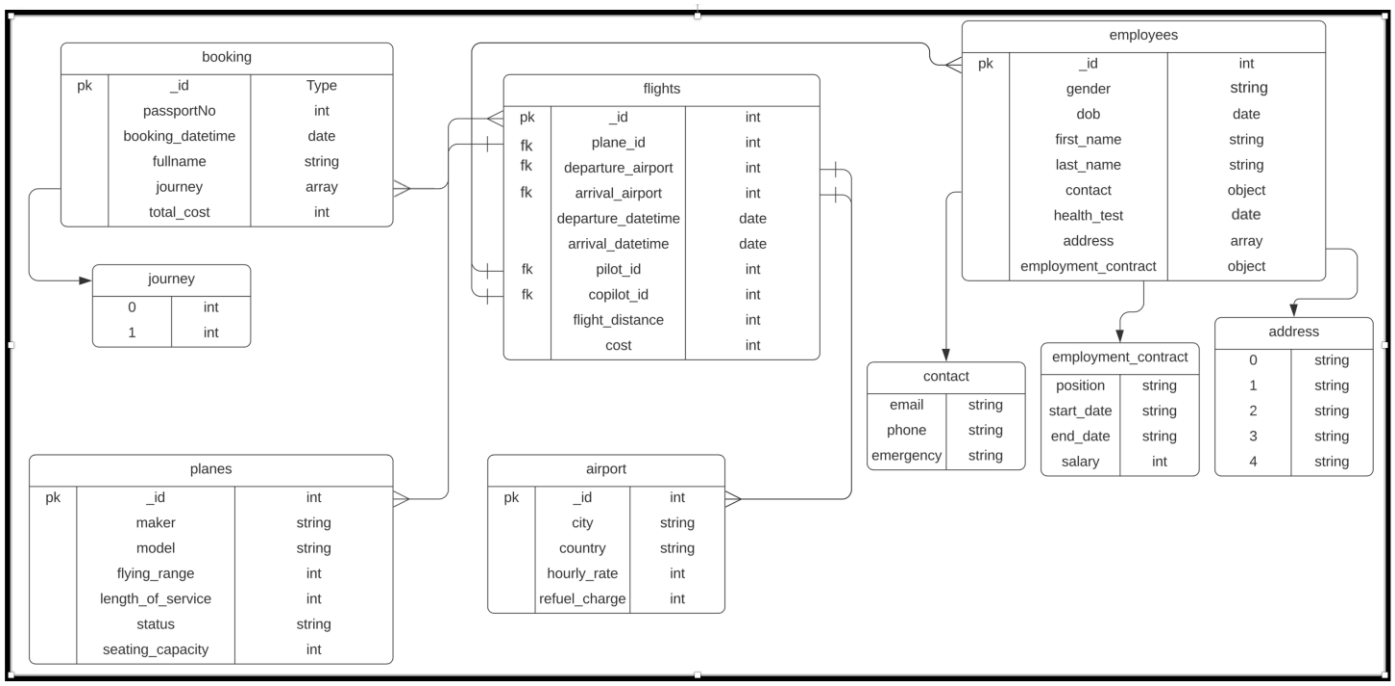


Figure 1: ER Diagram

The employee collection stores details of employees at the airline, there is a one-to-many relationship between an employee and their employment record. Employment details (employment_record) is stored as an embedded document. This allows any changes in the employment such as job title or salary to be stored in the database. The embedded document has the following fields: position, start_date, end_date, salary. The employee's current role can be found by finding the entry which has no end_date.

The address is stored as an array allowing the different components of the address (house number, street, etc) to be stored as different elements in the array.

Employees (employee)

- _id (int)
- address (array)
- contact (object/embedded document)
    - email (string)
    - emergency (string)
    - phone (string)
- dob (date)

- employment_record (object/embedded document)
    - position (string)
    - start_date (string)
    - end_date (string)
    - salary (int)
- first_name (string)
- gender (string)
- health_test (date)
- last_name (string)

The “planes” collection stores information about aircraft owned by the company. The collection has an _id field which acts as a unique identifier akin to a serial number. Additionally, it has a 1-to-1 relationship with the “flights” collection. This means that there can only be a single aircraft per flight.

There is additional data relevant to the aircraft, such as seating capacity, which could be used to ensure the aircraft cannot be overbooked. The flying range is useful in determining the most suitable aircraft for a specific flight. Lastly, length of service can be used to determine when an aircraft should be retired or if it’s very old and something breaks. In this case it might be easier for the company to replace the plane entirely.

Planes

- _id (int)
- flying_range (int)
- length_of_service (int)
- maker (string)
- model (string)
- seating_capacity (int)
- status (string)

The airports collection contains information about all airports that the airline runs flights between. The key information stored is the name and its location, which is split into its city and country. Since airports can change names or might even have the same name in different locations, they are uniquely identified using an “_id” field.

Additionally, the collection also stores information about charges the airport incurs. These are the hourly rate, which is a fixed rate charged to the company per aircraft and refuel charge, which is a fixed payment charged to the company to refuel an aircraft.

Lastly, this collection has a many-to-many relationship with flights and planes. The local field "_id" is referenced in the "flights" collection, namely in "departure_airport" and "arrival_airport" fields.

Airports

- _id (int)
- city (string)
- country (string)
- hourly_rate (int)
- name (string)
- refuel_charge (int)

The "flights" collection is by far one of the most important ones. It contains information about every flight the company runs. The collection stores information such as the "plane_id" of an aircraft used for a specific flight, departure and arrival dates and times, departure and arrival airport IDs, the pilot/co-pilot flying the aircraft, and flight distance.

This collection stores five different IDs as it references the "airports", "planes" and "employees" table. The relationship between "flights" and "airports" is one-to-many, as a single flight uses two airports as origin and destination points.

The relationship between "flights" and "planes" is one-to-one, as a single flight is conducted by a single aircraft. The collections "flights" and "employees" have a one-to-many relationship, as a single flight requires a pilot flying, and a first officer/co-pilot to fly the aircraft.

While we could embed one to one relationships such as pilot_id, copilot_id, departure_airport, arrival_airport, plane_id, we have instead chosen to reference the respective collections. Embedding would result in faster queries as we would not need to use lookups. However, there are several reasons for referencing:

- Some of our queries such as query 10 which finds the most expensive airport for a country are much easier to write using a separate airports collection. Doing a similar query using embedded airports would prove difficult.
- There would be a lot of repetition of key data as there is a one-to-many relationship between airport/employee and flights. E.g., one airport may have many flights. If this data was embedded, changing something like the hourly_rate of an airport would mean that many rows in flights would need to change. If this data was not changed correctly, this could lead to inconsistency where the hourly_rate for the same airport may be different for different flights presenting a significant problem.

Flights

- _id (int)
- arrival_airport (int)
- arrival_datetime (date)
- departure_airport (mixed – since it contains nulls and int)
- departure_datetime (date)
- pilot_id (mixed – since it contains null/int)
- copilot_id (mixed – since it contains null/int)
- flight_distance (mixed – since it contains null/int)
- plane_id (mixed – since it contains null/int)

The bookings collection contains information about every booking made by a passenger and includes their information. The primary information in this collection is the passenger's name, stored as "full_name", their passport number as "passportNo" and their journey, which is in fact an array of flight IDs, which would be either a one-to-one relationship, or a one-to-many, depending on the number of items in the array.

Since journey has a many to many relationship with flights, this is an example of a place where references are definitely much more suitable than embedding. The flight data itself is also quite complex with a lot of fields.

The final field is "total_cost", which is a sum of the individual flight costs. If the passenger has only booked a single flight, they will only be charged for that flight, otherwise, their booking cost will be reflected according to the total sum of flights booked.

Bookings

- _id (int)
- booking_datetime (date)
- full_name (string)
- journey (mixed – since it contains null/array)
- passportNo (mixed – since it contains null/int)
- total_cost (mixed – since it contains null/int)

C.Implementing and populating the Schema

A MongoDB file containing insert queries for our sample data is given as **initialize_db.mongodb** in the zip solder submitted. It includes a range of sample data for each collection implemented in our airline system. To show the robustness of some of the queries implanted we intentionally included some empty data to test the boundaries of our queries. This includes some bookings with no flights allocated and/or missing employee details. The list is not exhaustive and would need to be expanded to fit more real-world scenarios.

D.Queries

Below is a list of 12 MongoDB queries that extract information we deemed might be of interest to the airline. They explore a range of MongoDB constructs covered in the lectures including find and aggregate frameworks like $match, $project, $sort, $group, $unwind, $limit, $add, $subtract, $multiply, $sum, $count, $size, $addfields etc. Starting with simple find queries going up to complex queries that span most of the collections are implemented each with a real-world reasoning attached to it of how it may be used by the airline.

A brief explanation what a query does, and a text copy of the queries and a screenshot of expected output are provided under each query.

Query #1 Find all the planes that are working.

Find how many planes are in working condition. This is a basic query which is useful for knowing the total number of working aircraft available to the company at any time.

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

Expected Output:

```
9
```

Query #2 Find the gender demographic of the airline.

Find out if more males or females are being employed by the airline.

```
db.employees.find({gender: "M"}).count() > db.employees.find({gender: "F"}) ? "More Females Employees" : "More Male Employees";
```

Expected Output:

```
More Male Employees
```

Query #3 Top 3 flying passengers.

Find top 3 clients spending the most, using the unique passenger passport number, calculating total expenses and returning the 3 highest spenders along with their name and passport number after sorting them in descending order. This is often used by airlines to reward most loyal customers and for advertising purposes.

```
db.bookings.aggregate([
{
  $project: {
     booking_datetime: 0,
     journey: 0,
  }
},
{
  $group:{
  _id:"$passportNo",
  Passenger:{$first:"$fullname"},
  total_expense:{$sum:"$total_cost"}
  }
},
{
$addFields: { Passport: "$_id" }
},
{
$project: { _id: 0 }
},
{
  $sort: {total_expense: -1}
},
{
  $limit: 3
}
])
```

Expected Output:


```
[
  {
    "Passenger": "Angell, Norman",
    "total_expense": 1220,
    "passport": 268160
  },
  {
    "Passenger": "Angelou, Maya",
    "total_expense": 1150,
    "passport": 284847
  },
  {
    "Passenger": "Amiel, Barbara",
    "total_expense": 815,
    "passport": 123789
  }
]
```

Query #4 Pilot of the month

Find the pilot who has flown the greatest distance, left outer join the flights and employees collections and return the total number of flights and total distance flown by the top flying pilot along with their first name, last name, employee ID, and email to maybe award the pilot in a real-world application.

Query:

```
db.employees.aggregate(
  {
  $lookup:
          {
             from:"flights",
             localField:"_id",
             foreignField:"pilot_id",
             as:"flights"
          }
},
{
  $project:{
        _id:1,
        first_name:"$first_name",
        last_name:"$last_name",
        email:"$contact.email",
        no_of_flights:{$size:"$flights"},
        total_distance:{$sum:"$flights.flight_distance"}

     }
},
{ $sort : { total_distance : -1 } },
{ $limit : 1 }
)
```

Expected Output:

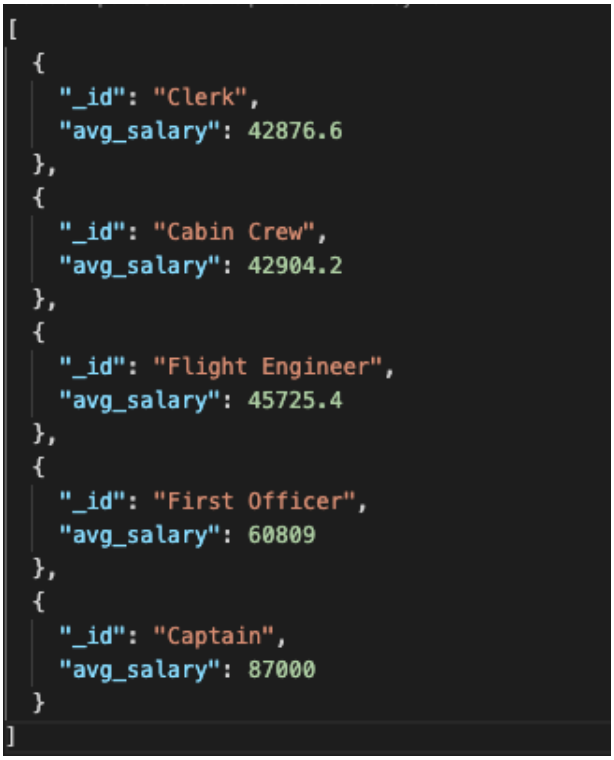
Query #5 Average salary by job title

This query will find the average salary for each job title in the employees collection. This has several applications, e.g.:

- To give an idea of what salary to offer when hiring for a job title.
- In performance reviews, to see whether to give an employee a raise based on how their salary compares with the average for their job title.
- To compare with industry averages.

```
db.employees.aggregate([
  {
     $unwind: "$employment_record"
  },
 {
    $group : {
    _id:"$employment_record.position",
    avg_salary:{$avg:"$employment_record.salary"}
  }},
  {
     $sort:{avg_salary:1}
  }
])
```

Expected Output:



```
[
  {
    "_id": "Clerk",
    "avg_salary": 42876.6
  },
  {
    "_id": "Cabin Crew",
    "avg_salary": 42904.2
  },
  {
    "_id": "Flight Engineer",
    "avg_salary": 45725.4
  },
  {
    "_id": "First Officer",
    "avg_salary": 60809
  },
  {
    "_id": "Captain",
    "avg_salary": 87000
  }
]
```

Query #6 Find the number of available seats for each plane

This query will find the number of seats booked by performing a look-up from flights to bookings on flights = bookings.journey. The total number of seats is found by performing a look-up from flights to planes on flights.plane_id = planes._id. Finally, the number of available seats is found using total_seats - booked_seats.

This query is useful for determining how many tickets can still be sold for a flight and for determining how much to charge for tickets. If there are lots of available seats, the airline could reduce the price of a ticket. If there are hardly any seats, they could increase the price.

```
db.flights.aggregate([
{
   $lookup:
           {
              from:"bookings",
              localField:"_id",
              foreignField:"journey",
              as:"flight_bookings"
           }
},
{
   $lookup:
           {
              from: "planes",
              localField: "plane_id",
              foreignField: "_id",
              as: "plane_specs"
           }
},
{
   $lookup:
           {
              from: "airports",
              localField: "departure_airport",
              foreignField: "_id",
              as: "departure"
           }
},
{
   $lookup:
           {
              from: "airports",
              localField: "arrival_airport",
              foreignField: "_id",
              as: "arrival"
           }
},
 {$unwind: '$plane_specs'},
 {$unwind: '$departure'},
{$unwind: '$arrival'},
{
   $project:{
         _id:1,
         departure: "$departure.name",
```

```
        departure_time: { $dateToString: { format: "%d-%m-%Y %H:%M", date: "$departure_datetime" } },
        arrival: "$arrival.name",
        arrival_time: { $dateToString: { format: "%d-%m-%Y %H:%M", date: "$arrival_datetime" } },
        total_bookings:{$size:"$flight_bookings"},
        total_seats : "$plane_specs.seating_capacity",
        available_seats: {$subtract:["$plane_specs.seating_capacity", {$size:"$flight_bookings"}]}
      }
}
])
```

Expected Output:

```
[
  {
    "_id": 1,
    "departure": "Heathrow",
    "departure_time": "15-12-2021 04:30",
    "arrival": "Birmingham Airport",
    "arrival_time": "15-12-2021 16:30",
    "total_bookings": 1,
    "total_seats": 155,
    "available_seats": 154
  },
  {
    "_id": 2,
    "departure": "Birmingham Airport",
    "departure_time": "15-12-2021 12:00",
    "arrival": "Ibiza Airport",
    "arrival_time": "16-12-2021 00:00",
    "total_bookings": 1,
    "total_seats": 190,
    "available_seats": 189
  },
```

Query #7 Find how much airports in the USA are generating as booking income

Find out the total amount of money the passengers spend on travelling in the USA.

```
db.airports.aggregate([
  { $match: { country : "USA" }
  },
  {
     $lookup:
    {
      from: "flights",
      localField: "_id",
      foreignField: "arrival_airport",
      as: "flights_docs"
    }
  },
  {
   $lookup:
    {
      from: "bookings",
      localField: "flights_docs._id",
      foreignField: "journey",
      as: "booking_docs"
    }
  },
```

```
  {   $unwind: "$booking_docs"
  },
  {
     $group: {
           _id: "$name",
           moneySpent: {$sum: "$booking_docs.total_cost"}
     }
  }
])
```

Expected Output:

```
[
  {
    "_id": "John F Kennedy International",
    "moneySpent": 1900
  },
  {
    "_id": "Dallas Airport",
    "moneySpent": 1580
  },
  {
    "_id": "Denver Airport",
    "moneySpent": 1130
  },
  {
    "_id": "Seattle Airport",
    "moneySpent": 3025
  }
]
```

Query #8 Print passenger ticket using passport number

Find out all the details using the passport number of the passenger that made the booking. These details include the name, journey, pilot, booking details etc. This query will be helpful to get all details of the passenger in case of an emergency and can only be used by the airlines for security reasons.

```
db.bookings.aggregate([
{
  $match:{  passportNo :123415}
},
{
  $lookup:
    {
      from: "flights",
      localField: "journey",
      foreignField: "_id",
      as: "flights"
    }
  },
  {
  $lookup:
    {
      from: "airports",
```

```
    localField: "flights.departure_airport",
    foreignField: "_id",
    as: "departure_airport"
   }
 },
 {
 $lookup:
   {
    from: "airports",
    localField: "flights.arrival_airport",
    foreignField: "_id",
    as: "arrival_airport"
   }
 },
 {
  $lookup:
   {
    from: "planes",
    localField: "flights.plane_id",
    foreignField: "_id",
    as: "planes"
   }
 },
 {
  $project:{
    passportNo:1,
    booking_datetime:1,
    fullname:1,
    total_cost:1,
    Flight_Stops:{$size:"$journey"},
    "flights.departure_datetime":1,
    "flights.arrival_datetime":1,
    "departure_airport.name":1,
    "departure_airport.city":1,
    "arrival_airport.name":1,
    "arrival_airport.city":1,
    "planes.maker":1,
    "planes.model":1,
  }
 }
])
```

Expected Output:

Query #9 Find any flights which have a distance that is too large

This query will perform a lookup to planes on flights.plane_id = planes._id. Entries from flights where flights.flight_distance > planes.flying_range are then returned, corresponding to flights where the plane would need to refuel mid-flight.

This query can be used when planning routes. In the example below, a flight from Sydney to Moscow is 15000km however the plane can only fly 6150km. This means either a plane with a higher-flying range would need to be used, or there would have to be multiple connecting flights.

```
db.flights.aggregate(
    {$lookup:
     {from: "planes",
      localField: "plane_id",
      foreignField: "_id",
      as: "plane_specs"
     }},
  {$unwind: '$plane_specs'},
   {$match:{$expr:{$gt:["$flight_distance", "$plane_specs.flying_range"]}}},
```

```
    {$lookup:
     {from: "airports",
      localField: "departure_airport",
      foreignField: "_id",
      as: "departure"
     }},
      {$lookup:
     {from: "airports",
      localField: "arrival_airport",
      foreignField: "_id",
      as: "arrival"
     }},
     {$unwind: '$departure'},
      {$unwind: '$arrival'},
     {$project:{
       "_id": 1,
       "departure": "$departure.city",
       "arrival": "$arrival.city",
       "distance": "$flight_distance",
       "plane_flying_range": "$plane_specs.flying_range"
     }}
  )
```

Expected Output:

```
[
  {
    "_id": 20,
    "departure": "Sydney",
    "arrival": "Moscow",
    "distance": 15000,
    "plane_flying_range": 6150
  }
]
```

Query #10 Most expensive airport by country

Gets the most expensive airport per country based on hourly charge to the company.

```
// get most expensive airport by country
db.airports.aggregate([
  {
     $sort:
        {
           hourly_rate: -1
        }
  },
  {
     $group:
        {
           _id: "$country",
           highestCharge:
        {
           $max: "$hourly_rate"
        },
           airport:
              {
```

```
                $first: "$name"
            }
        }
    },
]);
```

Expected Output:

Include Import Statements | Include Driver Syntax


```
[
  {
    "_id": "UK",
    "highestCharge": 135,
    "airport": "Edinburgh Airport"
  },
  {
    "_id": "India",
    "highestCharge": 221,
    "airport": "IGI New Delhi"
  },
  {
    "_id": "Australia",
    "highestCharge": 100,
    "airport": "Sydney Airport"
  },
  {
    "_id": "France",
    "highestCharge": 238,
    "airport": "Bordeaux Airport"
  },
  {
    "_id": "UAE",
    "highestCharge": 204,
    "airport": "Abu Dhabi International"
  },
  {
    "_id": "Spain",
    "highestCharge": 115,
    "airport": "Ibiza Airport"
  },
  {
    "_id": "USA",
    "highestCharge": 229,
    "airport": "Dallas Airport"
  },
  {
    "_id": "Russia",
    "highestCharge": 120,
    "airport": "Sheremetyevo imeni A. S. Pushkina"
  }
]
```

Query #11 Calculate age of employees.

Query finds all pilots (staff employed as captains and first officers) and calculates their current age by subtracting their date of birth from today's date.

```
db.employees.aggregate([
  {
    $match: {
      $or:
        [
          {
            "employment_record.position": "Captain"
          },
          {
            "employment_record.position": "First Officer"
          }
        ]
    }
  },
  {
    $sort:
      {
        last_name: -1
      }
  },
  {
    $project:
      {
        _id: "$_id",
        name: "$last_name",
        age:
          {
            $dateDiff:
              {
                startDate: "$dob",
                endDate: "$$NOW",
                unit: "year"
              }
          },
        position: "$employment_record.position"
      }
  }
]);
```

Expected Output:

Query #12 Find the net revenue generated by the airlines

Find what the total net revenue of the company is, that is if it is in loss or gain. This query will be calculated by total income subtracted by total expense and salary to the employee. This query will give the idea of how the company is doing and how it can increase its revenue.

Since this query will actually need to get data from 3 separate collections i.e., employees (for salary), bookings (for income), and flights (for flying costs), 3 separate queries were written, and the results were saved to variables.

flight_cost determines the cost of running the flights. A lookup to the airport is done on flights.departure_airport = airport._id. As mentioned in our assumptions, we are assuming the refuel_charge to be the same for all planes at an airport and we are also assuming each plane before departing would have waited at the airport for an hour. The cost to fly can then be found by simply doing airport.refuel_charge + airport.hourly_rate. In order to not double count the costs of flying (e.g., if a plane goes from airport A -> B -> C, the costs from B would be counted twice if calculating costs for departure and arrival), this calculation is only done on the departure airport. Our final assumption is that when not in use planes can be stored for free.

Salary determines the cost of employing the airline staff. As mentioned in our assumptions, we are assuming the data in the database is over a one-month timespan and that employees work their normal hours. This means the salary can be found by summing together the salaries and dividing them by 12 to get a month's salary.

booking_income determines the income raised from selling tickets. This is simply the sum of the bookings.total_cost. Our bookings collection only has 15 entries. Adding more entries to reflect the number of bookings more accurately on a real flight would be time-consuming so instead the booking_income is multiplied by 100.

Finally, the revenue can be found using booking_income - flight_cost - salary

Query:

```
var flight_cost = db.flights.aggregate(
 {
   // Join airports to get airport charges
   $lookup:
     {from: "airports",
      localField: "departure_airport",
      foreignField: "_id",
      as: "airport"
```

```
      }
  },
  // Unwind airport array
  {$unwind: '$airport'},
  {$group: {
        _id: null,
        // Add refuel_charge and hourly_rate
        total: {$sum: { $add: ['$airport.refuel_charge', '$airport.hourly_rate']}}}},
  {$project: {_id: 0, total:1}}).toArray()[0]["total"]

var salary = db.employees.aggregate(
  // Unwind employment_record embedded document
  {$unwind: '$employment_record'},
  {$group: {
    _id: null,
    // Divide salary by 12 to get monthly salary
    salary: {$sum: { $multiply: [ "$employment_record.salary", (1/12) ] }}
  }}
).toArray()[0]["salary"] // Convert result to a value

var booking_income = db.bookings.aggregate(
  {$group: {
    _id: null,
    // Sum total_cost to get total_income
    income: {$sum: "$total_cost"}}},
  {$project: {
    _id:0,
    // Multiply income by 100 to model there being more bookings
    income:{$multiply:["$income", 100]
  }}}
).toArray()[0]["income"]

//Calculate revenue
booking_income - flight_cost - salary
```

Expected Output:

A revenue of £733570 was initially obtained.

Part Two

A. Explain and Indexing

The query output of the explain function was analysed using the built-in MongoDB explain utility that analyses query execution statistics including the query plans as a tree of stages, execution time in milliseconds and number of documents examined, thus providing a useful insight when attempting to optimize a query. The output result of the Explain utility is too long to include here and hence have been combined in JavaScript file separately and submitted. Informative sections are included below and explained in detail.

Query 8 (Printing a passenger ticket) Explainer output.

Since this print passenger ticket query uses the $match stage at the beginning of the pipeline we can make use of an index to filter documents faster and easier. The $lookup stages that follow merge data using left outer joins to various collections. Unfortunately, MongoDB does not allow us to use indexes for lookups and hence any noticeable improvement is not expected in these lookups.

As we can see in the small snippet of result below, the explain utility was initially called without any additional indexes added to our collections. And as expected the document had to go through all collections hence “COLLSCAN” in the input stage and the total documents examined is 15 (equal to the size of the sample data given) as it had to go through the entire documents in the collections before returning the output of the match pipeline. However, after adding index for the passport number the key, we performed match on the input stage switched to IXSCAN with only one document examined by scanning index keys rather than the whole document. Likewise, the total execution time was reduced significantly from 1 millisecond to as low as 0 milliseconds.

Without Index:


```
"inputStage": {
          "stage": "COLLSCAN",
          "filter": {
            "passportNo": {
              "$eq": 123415
            }
          },
          "direction": "forward"
"executionStats": {
       "executionSuccess": true,
       "nReturned": 1,
       "executionTimeMillis": 1,
       "totalKeysExamined": 0,
```

```
        "totalDocsExamined": 15,
```

With Index:

```
"inputStage": {
          "stage": "FETCH",
          "inputStage": {
           "stage": "IXSCAN",
           "keyPattern": {
            "passportNo": 1
           },
           "indexName": "passportNo",
"executionStats": {
       "executionSuccess": true,
       "nReturned": 1,
       "executionTimeMillis": 0,
       "totalKeysExamined": 1,
       "totalDocsExamined": 1,
```

Query 11 (Calculating the age of employees) Explainer output.

This query was chosen because it uses $match and $sort which should benefit from the use of indexes as we are not returning the entire collection in our query like with some of our other queries. In cases where we are returning the entire collection, indexing has less of an impact.

After running db.employees.aggregate(...).explain('executionStats') without any indexes the following results were obtained. Since the output was 203 lines, the output was cut down to only include fields that will be discussed.

```
"executionStats": {
   "nReturned": 15,
   "executionTimeMillis": 0,
   "totalKeysExamined": 0,
   "totalDocsExamined": 35,
   "executionStages": {
        "stage": "SORT",
        "inputStage": {
         "stage": "COLLSCAN",
```

Due to the low number of rows (35), the execution time is already 0 milliseconds. However, we can see that MongoDB is performing a “COLLSCAN” when sorting. This means that the entire employees’ collection must be read when sorting which makes this query not very scalable as if there were a large number of employees in the collection this query could take a long time.

Indexes were then added to employees for employment_record.position (used for $match) and last_name (used for $sort). This query was run, and the following output was obtained.

```
"executionStats": {
   "nReturned": 15,
   "executionTimeMillis": 0,
   "totalKeysExamined": 35,
   "totalDocsExamined": 35,
   "executionStages": {
      "stage": "FETCH",
```

```
"stage": "IXSCAN",
```

As seen, we are now using a FETCH and IXSCAN instead of a COLLSCAN. This, in theory, should be faster, however, to confirm this a larger dataset would be needed to show significant improvement.

We have also experimented with adding additional indexes for the other fields of the collection and analysed the effect, it was observed there was no noticeable improvement in performance since some of those indexes are not used in the queries unless there is a constructor that makes use of the index like $match.

B. Profiling

MongoDB comes with a profiling utility that can be used to monitor and tune the performance of the queries performed on documents of a collection. This can be used to identify slow running queries, error rates and stack traces. It works by collecting query throughput, execution performance and buffer metrics. Profiling output of six queries were performed and their successive output is included.

Plan Summary if COLLSCAN for most of the queries profiled as it collectively examined every document of the collection. Similarly, we have zero the number of key examined which can be a good indication that our queries are performant.

The execution time in milliseconds was 0ms for our queries since we have a small sample data that is being tested by our queries.

General Output Descriptions on using aggregate.

Following lines are a step-by-step explanation of what steps the database engine took to process the query.

Runs COLLSCAN, takes 0ms to complete, aggregate bookings
Runs IXSCAN, takes 0ms to complete, findAndModify bookings
Runs COLLSCAN, takes 0ms to complete, aggregate bookings
Takes 2ms to complete, collStats bookings
Takes 0ms to complete, aggregate bookings
Takes 0ms to complete, listIndexes bookings
Takes 0ms to complete, aggregate bookings
Runs COUNT_SCAN, takes 0ms to complete, aggregate bookings
Runs COLLSCAN, takes 0ms to complete, find bookings
Runs COLLSCAN, takes 0ms to complete, aggregate bookings

The profiling output was produced by “db.system.profiler” command, we used a sort filter to sort the output by timestamp in descending order. We have also limited the total results to 10 initially, but then reduced to 2. Finally, we set the “slowms” parameter to 0 as our query runtimes appear to

be between 0 and 1 milliseconds. We experimented with different values of the parameters of the MongoDB profiler utility and to our surprise we could not observe much difference.

Profiling output for Query 3


```
[
  {
    "op": "command",
    "ns": "airline.bookings",
    "command": {
      "aggregate": "bookings",
      "pipeline": [
        {
          "$project": {
            "booking_datetime": 0,
            "journey": 0
          }
        },
        {
          "$group": {
            "_id": "$passportNo",
            "Passenger": {
              "$first": "$fullname"
            },
            "total_expense": {
              "$sum": "$total_cost"
            }
          }
        },
        {
          "$addFields": {
            "Passport": "$_id"
          }
        },
        {
          "$project": {
            "_id": 0
          }
        },
        {
          "$sort": {
            "total_expense": -1
          }
        },
        {
          "$limit": 3
        }
      ],
      "cursor": {},
      "lsid": {
        "id": {
          "$binary": {
            "base64": "N1vhrxFQTemsK93bCWwadw==",
```

```
            "subType": "04"
          }
        }
      },
      "$db": "airline"
    },
    "keysExamined": 0,
    "docsExamined": 15,
    "hasSortStage": true,
    "cursorExhausted": true,
    "numYield": 0,
    "nreturned": 3,
    "locks": {
      "ReplicationStateTransition": {
        "acquireCount": {
          "w": 1
        }
      },
      "Global": {
        "acquireCount": {
          "r": 3
        }
      },
      "Mutex": {
        "acquireCount": {
          "r": 2
        }
      }
    },
    "flowControl": {},
    "storage": {},
    "responseLength": 338,
    "protocol": "op_msg",
    "millis": 0,
    "planSummary": "COLLSCAN",
    "ts": {
      "$date": "2021-12-23T16:14:50.516Z"
    },
    "client": "172.17.0.1",
    "appName": "mongodb-vscode 0.7.0",
    "allUsers": [],
    "user": ""
  }
]
```


Profiler output for Query 5


```
[
  {
    "op": "command",
    "ns": "airline.employees",
    "command": {
      "aggregate": "employees",
```

```
  "pipeline": [
    {
      "$unwind": "$employment_contract"
    },
    {
      "$group": {
        "_id": "$employment_contract.position",
        "avg_salary": {
          "$avg": "$employment_contract.salary"
        }
      }
    },
    {
      "$sort": {
        "avg_salary": 1
      }
    }
  ],
  "cursor": {},
  "lsid": {
    "id": {
      "$binary": {
        "base64": "2vlrBcTfQrKsrMfQpkiXCA==",
        "subType": "04"
      }
    }
  },
  "$db": "airline"
},
"keysExamined": 0,
"docsExamined": 35,
"hasSortStage": true,
"cursorExhausted": true,
"numYield": 0,
"nreturned": 5,
"locks": {
  "ReplicationStateTransition": {
    "acquireCount": {
      "w": 1
    }
  },
  "Global": {
    "acquireCount": {
      "r": 3
    }
  },
  "Mutex": {
    "acquireCount": {
      "r": 2
    }
  }
},
"flowControl": {},
```

```
    "storage": {},
    "responseLength": 346,
    "protocol": "op_msg",
    "millis": 0,
    "planSummary": "COLLSCAN",
    "ts": {
      "$date": "2021-12-23T16:15:34.859Z"
    },
    "client": "172.17.0.1",
    "appName": "mongodb-vscode 0.7.0",
    "allUsers": [],
    "user": ""
  }
]
```


Profiler Output for Query 8


```
[
  {
    "op": "command",
    "ns": "airline.bookings",
    "command": {
      "aggregate": "bookings",
      "pipeline": [
        {
          "$match": {
            "passportNo": 123415
          }
        },
        {
          "$lookup": {
            "from": "flights",
            "localField": "journey",
            "foreignField": "_id",
            "as": "flights"
          }
        },
        {
          "$lookup": {
            "from": "airports",
            "localField": "flights.departure_airport",
            "foreignField": "_id",
            "as": "departure_airport"
          }
        },
        {
          "$lookup": {
            "from": "airports",
            "localField": "flights.arrival_airport",
            "foreignField": "_id",
            "as": "arrival_airport"
          }
        },
        {
```

```
      "$lookup": {
        "from": "planes",
        "localField": "flights.plane_id",
        "foreignField": "_id",
        "as": "planes"
      }
    },
    {
      "$project": {
        "passportNo": 1,
        "booking_datetime": 1,
        "fullname": 1,
        "total_cost": 1,
        "Flight_Stops": {
          "$size": "$journey"
        },
        "flights.departure_datetime": 1,
        "flights.arrival_datetime": 1,
        "departure_airport.name": 1,
        "departure_airport.city": 1,
        "arrival_airport.name": 1,
        "arrival_airport.city": 1,
        "planes.maker": 1,
        "planes.model": 1
      }
    }
  ],
  "cursor": {},
  "lsid": {
    "id": {
      "$binary": {
        "base64": "bv/I3yoZRUyLHjySoiBKPQ==",
        "subType": "04"
      }
    }
  },
  "$db": "airline"
},
"keysExamined": 4,
"docsExamined": 19,
"cursorExhausted": true,
"numYield": 0,
"nreturned": 1,
"queryHash": "A300CFDE",
"planCacheKey": "D6154327",
"locks": {
  "ReplicationStateTransition": {
    "acquireCount": {
      "w": 1
    }
  },
  "Global": {
    "acquireCount": {
```

```
        "r": 11
       }
      },
      "Mutex": {
       "acquireCount": {
        "r": 10
       }
      }
     },
     "flowControl": {},
     "storage": {},
     "responseLength": 567,
     "protocol": "op_msg",
     "millis": 1,
     "planSummary": "COLLSCAN",
     "ts": {
      "$date": "2021-12-23T16:17:12.871Z"
     },
     "client": "172.17.0.1",
     "appName": "mongodb-vscode 0.7.0",
     "allUsers": [],
     "user": ""
   }
]
```


Profiler Output for Query 6


```
[
  {
    "op": "command",
    "ns": "airline_group_22.flights",
    "command": {
      "aggregate": "flights",
      "pipeline": [
        {
          "$lookup": {
            "from": "bookings",
            "localField": "_id",
            "foreignField": "journey",
            "as": "flight_bookings"
          }
        },
        {
          "$lookup": {
            "from": "planes",
            "localField": "plane_id",
            "foreignField": "_id",
            "as": "plane_specs"
```

```
  }
},
{
  "$lookup": {
    "from": "airports",
    "localField": "departure_airport",
    "foreignField": "_id",
    "as": "departure"
  }
},
{
  "$lookup": {
    "from": "airports",
    "localField": "arrival_airport",
    "foreignField": "_id",
    "as": "arrival"
  }
},
{
  "$unwind": "$plane_specs"
},
{
  "$unwind": "$departure"
},
{
  "$unwind": "$arrival"
},
{
  "$project": {
    "_id": 1,
    "departure": "$departure.name",
    "departure_time": {
      "$dateToString": {
        "format": "%d-%m-%Y %H:%M",
        "date": "$departure_datetime"
      }
    },
    "arrival": "$arrival.name",
    "arrival_time": {
      "$dateToString": {
        "format": "%d-%m-%Y %H:%M",
        "date": "$arrival_datetime"
      }
    },
    "total_bookings": {
      "$size": "$flight_bookings"
```

```
        },
        "total_seats": "$plane_specs.seating_capacity",
        "available_seats": {
          "$subtract": [
            "$plane_specs.seating_capacity",
            {
              "$size": "$flight_bookings"
            }
          ]
        }
      }
    }
  ],
  "cursor": {},
  "lsid": {
    "id": {
      "$binary": {
        "base64": "sQLy9J6GQ5u4THrjmW2Jtw==",
        "subType": "04"
      }
    }
  },
  "$db": "airline_group_22"
},
"keysExamined": 47,
"docsExamined": 303,
"cursorExhausted": true,
"numYield": 0,
"nreturned": 15,
"queryHash": "A300CFDE",
"planCacheKey": "D6154327",
"locks": {
  "ReplicationStateTransition": {
    "acquireCount": {
      "w": 1
    }
  },
  "Global": {
    "acquireCount": {
      "r": 131
    }
  },
  "Mutex": {
    "acquireCount": {
      "r": 130
    }
```

```
      }
    },
    "flowControl": {},
    "storage": {},
    "responseLength": 3313,
    "protocol": "op_msg",
    "millis": 14,
    "planSummary": "COLLSCAN",
    "ts": {
      "$date": "2021-12-23T16:59:05.913Z"
    },
    "client": "127.0.0.1",
    "appName": "mongodb-vscode 0.7.0",
    "allUsers": [],
    "user": ""
  }
]
```


Profiler Output for Query 11


```
[
  {
    "op": "command",
    "ns": "airline_group_22.employees",
    "command": {
      "aggregate": "employees",
      "pipeline": [
        {
          "$match": {
            "$or": [
              {
                "employment_record.position": "Captain"
              },
              {
                "employment_record.position": "First Officer"
              }
            ]
          }
        },
        {
          "$sort": {
            "last_name": -1
          }
        },
        {
          "$project": {
            "_id": "$_id",
            "name": "$last_name",
            "age": {
```

```
          "$dateDiff": {
            "startDate": "$dob",
            "endDate": "$$NOW",
            "unit": "year"
          }
        },
        "position": "$employment_record.position"
      }
    }
  ],
  "cursor": {},
  "lsid": {
    "id": {
      "$binary": {
        "base64": "kgfIcVKJTv+ECLiSmAQ2wg==",
        "subType": "04"
      }
    }
  },
  "$db": "airline_group_22"
},
"keysExamined": 0,
"docsExamined": 35,
"hasSortStage": true,
"cursorExhausted": true,
"numYield": 0,
"nreturned": 19,
"queryHash": "175F6760",
"planCacheKey": "3F3188DE",
"locks": {
  "ReplicationStateTransition": {
    "acquireCount": {
      "w": 1
    }
  },
  "Global": {
    "acquireCount": {
      "r": 3
    }
  },
  "Mutex": {
    "acquireCount": {
      "r": 2
    }
  }
},
"flowControl": {},
"storage": {},
"responseLength": 1487,
"protocol": "op_msg",
"millis": 1,
"planSummary": "COLLSCAN",
"ts": {
```

```
      "$date": "2021-12-23T17:20:31.143Z"
    },
    "client": "127.0.0.1",
    "appName": "mongodb-vscode 0.7.0",
    "allUsers": [],
    "user": ""
  }
]
```


Profiler Output for Query 12


```
[
  {
    "op": "command",
    "ns": "airline.flights",
    "command": {
      "aggregate": "flights",
      "pipeline": [
        {
          "$lookup": {
            "from": "airports",
            "localField": "departure_airport",
            "foreignField": "_id",
            "as": "airport"
          }
        },
        {
          "$unwind": "$airport"
        },
        {
          "$group": {
            "_id": null,
            "total": {
              "$sum": {
                "$add": [
                  "$airport.refuel_charge",
                  "$airport.hourly_rate"
                ]
              }
            }
          }
        },
        {
          "$project": {
            "_id": 0,
            "total": 1
          }
        }
      ],
      "cursor": {},
      "lsid": {
        "id": {
```

```
        "$binary": {
          "base64": "yDLzRv3KSNe7B1JfmgZYJA==",
          "subType": "04"
        }
      }
    },
    "$db": "airline"
  },
  "keysExamined": 16,
  "docsExamined": 32,
  "cursorExhausted": true,
  "numYield": 0,
  "nreturned": 1,
  "queryHash": "A300CFDE",
  "planCacheKey": "D6154327",
  "locks": {
    "ReplicationStateTransition": {
      "acquireCount": {
        "w": 1
      }
    },
    "Global": {
      "acquireCount": {
        "r": 35
      }
    },
    "Mutex": {
      "acquireCount": {
        "r": 34
      }
    }
  },
  "flowControl": {},
  "storage": {},
  "responseLength": 123,
  "protocol": "op_msg",
  "millis": 1,
  "planSummary": "COLLSCAN",
  "ts": {
    "$date": "2021-12-23T17:16:05.031Z"
  },
  "client": "172.17.0.1",
  "appName": "mongodb-vscode 0.7.0",
  "allUsers": [],
  "user": ""
},
{
  "op": "command",
  "ns": "airline.employees",
  "command": {
    "aggregate": "employees",
    "pipeline": [
      {
```

```
      "$group": {
        "_id": null,
        "salary": {
          "$sum": "$employment_contract.salary"
        }
      }
    },
    {
      "$project": {
        "salary": {
          "$divide": [
            "$salary",
            12
          ]
        }
      }
    }
  ],
  "cursor": {},
  "lsid": {
    "id": {
      "$binary": {
        "base64": "yDLzRv3KSNe7B1JfmgZYJA==",
        "subType": "04"
      }
    }
  },
  "$db": "airline"
},
"keysExamined": 0,
"docsExamined": 35,
"cursorExhausted": true,
"numYield": 0,
"nreturned": 1,
"locks": {
  "ReplicationStateTransition": {
    "acquireCount": {
      "w": 1
    }
  },
  "Global": {
    "acquireCount": {
      "r": 3
    }
  },
  "Mutex": {
    "acquireCount": {
      "r": 2
    }
  }
},
"flowControl": {},
"storage": {},
```

```
  "responseLength": 135,
  "protocol": "op_msg",
  "millis": 0,
  "planSummary": "COLLSCAN",
  "ts": {
    "$date": "2021-12-23T17:16:05.035Z"
  },
  "client": "172.17.0.1",
  "appName": "mongodb-vscode 0.7.0",
  "allUsers": [],
  "user": ""
},
{
  "op": "command",
  "ns": "airline.bookings",
  "command": {
    "aggregate": "bookings",
    "pipeline": [
      {
        "$group": {
          "_id": null,
          "income": {
            "$sum": "$total_cost"
          }
        }
      },
      {
        "$project": {
          "_id": 0,
          "income": {
            "$multiply": [
              "$income",
              100
            ]
          }
        }
      }
    ],
    "cursor": {},
    "lsid": {
      "id": {
        "$binary": {
          "base64": "yDLzRv3KSNe7B1JfmgZYJA==",
          "subType": "04"
        }
      }
    },
    "$db": "airline"
  },
  "keysExamined": 0,
  "docsExamined": 15,
  "cursorExhausted": true,
  "numYield": 0,
```

```
    "nreturned": 1,
    "locks": {
      "ReplicationStateTransition": {
        "acquireCount": {
          "w": 1
        }
      },
      "Global": {
        "acquireCount": {
          "r": 3
        }
      },
      "Mutex": {
        "acquireCount": {
          "r": 2
        }
      }
    },
    "flowControl": {},
    "storage": {},
    "responseLength": 125,
    "protocol": "op_msg",
    "millis": 0,
    "planSummary": "COLLSCAN",
    "ts": {
      "$date": "2021-12-23T17:16:05.037Z"
    },
    "client": "172.17.0.1",
    "appName": "mongodb-vscode 0.7.0",
    "allUsers": [],
    "user": ""
  }
]
```