COURSEWORK 2 DATABASE DEVELOPMENT AND PERFORMANCE TUNING

CW2 -ECS789P_Group 28

Name: Minh Duc Nguyen

Student ID: 210740983

Contents

Part 1

1.1 Assumptions:

Below is the list of key assumptions for the coursework, data model, and each collection:

General:

- All _id field of the documents is auto-generated objectId type. The purpose of using this data type is to take into consideration operationalizing this data model, an objectId field containing the timestamp of the document was added. Furthermore, this field would assure the uniqueness of the ID in each document throughout all nodes in the cluster (in case of the need to scale up the system horizontally). With the mentioned two properties, using objectId type greatly enhances the data governance process.
- A well-operated system strongly emphasizes data quality problems that could occur from operational risk by human input on which will impact business decisions. In this data model, there are some built-in data validation rules for several fields. However, realizing that to make a successful data model with good data quality, the will be a need for cross-validation between fields, collections (e.g: all the cabin crew's employee ID of the flight collection needs to be in the employee collection…). For the scope of data modelling and performance tuning in this coursework, assuming that aside from server-side data validator, there will be application-side data validator too.
- All the data in this data model is dummy data for the sake of demonstration. In the operationalized system, there will be a need for some aggregation pipelines for computing some calculated fields (e.g: total cost of a flight, total booked seat, total revenue of a journey…)
- There will be a logging mechanism on the application side to handle exceptions from the data validation rules of this data model.
- This data model is a part of a galaxy schema design. Therefore, it includes three fact collections flight, booking, and journey.
- This data model is normalized assuming that the counterpart denormalized data model's performance does not outweigh it considering the cost of data duplication in this business use case.
- All the date-time field in this data model is in ISO format.
- There will be no MapReduce query demonstration, as of MongoDB 5.0 the map-reduce operation is deprecated. The reason is due to an aggregation pipeline provides better performance and usability than a map-reduce operation. (Ref)

Employee collection:

- This data model is built for a UK airplane company. Therefore, the field citizenship only has a data validator accepting two values ['UK', 'Non-UK']

Flight collection:

- Each flight only has up to two pilots, including the main pilot and a co-pilot.
- In the operationalized system, there will be an aggregation pipeline calculating the total cost of the flight from the hourly rate of the cabin crew including pilots.

Booking collection:

- Assuming that the _id of the booking collection could link to the customer entity managing the customers' data in which is not included in the scope of this coursework.

Journey collection:

- In the operationalized system, there will be an aggregation pipeline calculating the total revenue of the journey from the total fee from bookings, the total cost of flights, and the total cost for the transit airport.

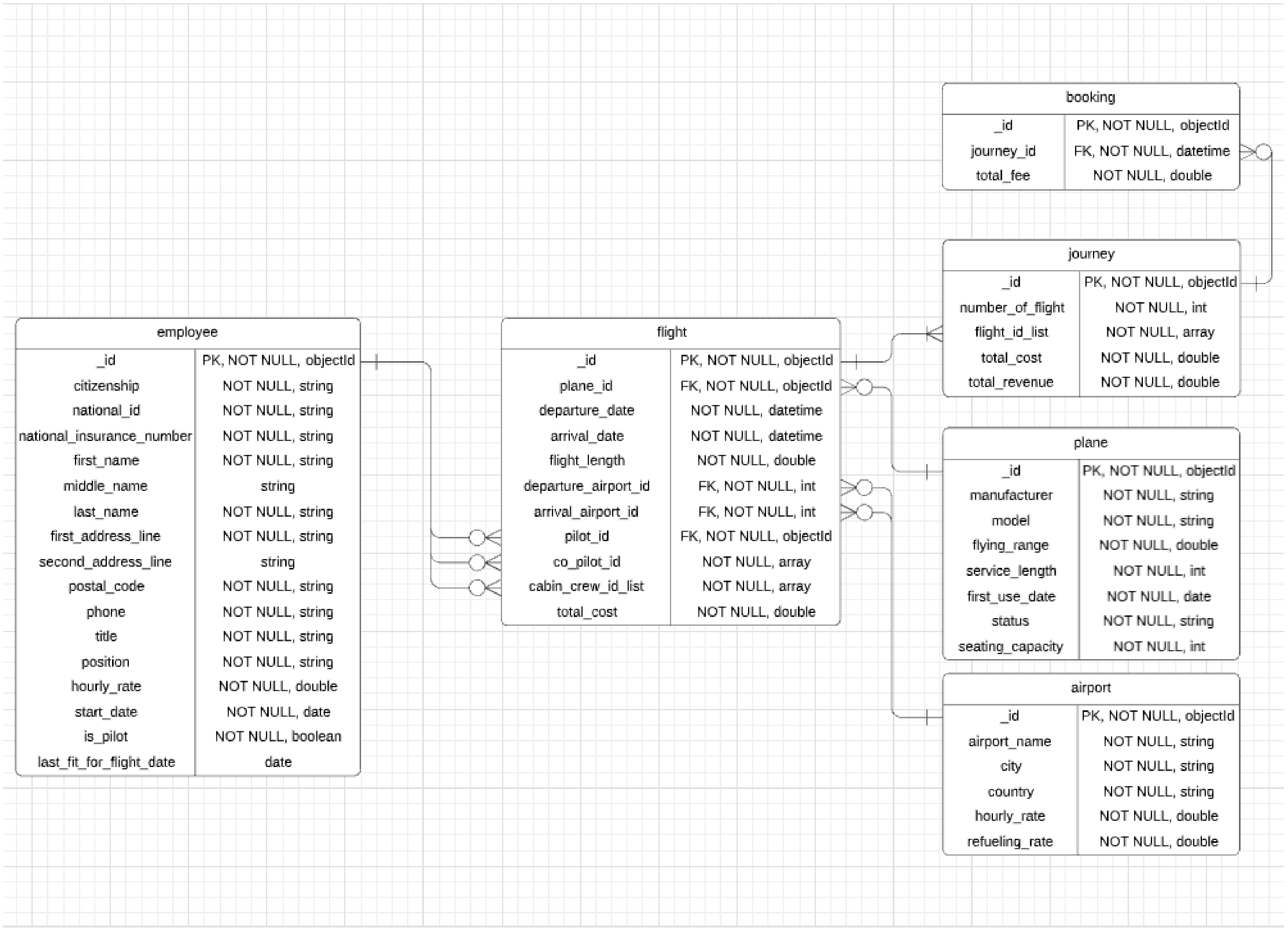
Plane collection:

- There are only three statuses of a plane in which reflect on the data validator of the script creating this collection.
- The will be no planned flights on the data for planes with the status “Being repaired” or “Upgrading”.

1.2 Schema design:

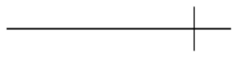
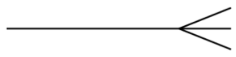
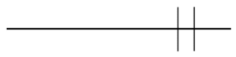
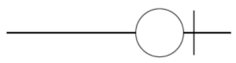
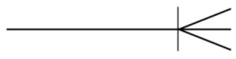
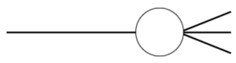
Entity-Relationship Diagram:

This coursework folder contains a data_model.png file with the below diagram in higher resolution.


booking
_id | PK, NOT NULL, objectId
journey_id | FK, NOT NULL, datetime
total_fee | NOT NULL, double
journey
_id | PK, NOT NULL, objectId
number_of_flight | NOT NULL, int
flight_id_list | NOT NULL, array
total_cost | NOT NULL, double
total_revenue | NOT NULL, double
employee
_id | PK, NOT NULL, objectId
citizenship | NOT NULL, string
national_id | NOT NULL, string
national_insurance_number | NOT NULL, string
first_name | NOT NULL, string
middle_name | string
last_name | NOT NULL, string
first_address_line | NOT NULL, string
second_address_line | string
postal_code | NOT NULL, string
phone | NOT NULL, string
title | NOT NULL, string
position | NOT NULL, string
hourly_rate | NOT NULL, double
start_date | NOT NULL, date
is_pilot | NOT NULL, boolean
last_fit_for_flight_date | date
flight
_id | PK, NOT NULL, objectId
plane_id | FK, NOT NULL, objectId
departure_date | NOT NULL, datetime
arrival_date | NOT NULL, datetime
flight_length | NOT NULL, double
departure_airport_id | FK, NOT NULL, int
arrival_airport_id | FK, NOT NULL, int
pilot_id | FK, NOT NULL, objectId
co_pilot_id | NOT NULL, array
cabin_crew_id_list | NOT NULL, array
total_cost | NOT NULL, double
plane
_id | PK, NOT NULL, objectId
manufacturer | NOT NULL, string
model | NOT NULL, string
flying_range | NOT NULL, double
service_length | NOT NULL, int
first_use_date | NOT NULL, date
status | NOT NULL, string
seating_capacity | NOT NULL, int
airport
_id | PK, NOT NULL, objectId
airport_name | NOT NULL, string
city | NOT NULL, string
country | NOT NULL, string
hourly_rate | NOT NULL, double
refueling_rate | NOT NULL, double

Side note and notation:

- This ER is crow's foot ER diagram.
- All the data type is according to mongoDB data type, except for datetime in which all date type in mongoDB is datetime already. This usage is just for illustrating the data type more clearly.
- PK: Primary Key
- FK: Foreign Key
- NOT NULL: the field is required to have value (with data validator)
- Crow's foot relationship notation:

Symbol	Meaning	Number
	One	N/A
	Many	N/A
	Mandatory-One	Exactly one
	Optional-One	Zero or one
	Mandatory-Many	One or More
	Optional-Many	Zero or more

-

Collections

Collection specifications, fields' data type & description, and validation rules will be included in the implementation script.

Below is the summary of relationships in the data model in the business use case context:

Employee

The _id field of the **employee** collection has a One-to-Optional-Many relationship with pilot_id, co_pilot_id, and value in the cabin_crew_id_list in the **flight** collection. This relationship infers that every pilot/crewmember could be in many flights before (not at the same time), or in the case of a new employee, there will be a pilot/crewmember that has not been in a flight yet.

Flight

The _id field of the **flight** collection has a One-to-Mandatory-Many relationship with the value in the flight_id_list in the **journey** collection. This relationship infers that every flight will be included in a journey for cost and revenue calculation. Furthermore, a flight could be in many journeys (e.g: stand-alone journeys and journeys that require transit).

Journey

The _id field of the **journey** collection has a One-to-Optional-Many relationship with journey_id in the **booking** collection. This relationship infers that every journey could have multiple different bookings or have not had booking yet.

Plane

The _id field of the **plane** collection has a One-to-Optional-Many relationship with plane_id in the **flight** collection. This relationship infers that every plane could have multiple different flights throughout time (not at the same time). However, there could be a case of a newly procured airplane that has not had flight yet.

Airport

The _id field of the **airport** collection has a One-to-Optional-Many relationship with departure_airport_id and arrival_airport_id in the **flight** collection. This relationship infers that every airport could be the departure or arrival point for none or multiple flights.

1.3 Schema implementation code:

This coursework folder contains the below code in the schema_implementation_code.js script.

```
// Create employee collection with validator on data type and value
db.createCollection("employee", {
  validator: {
    $jsonSchema: {
      bsonType: 'object',
      required: [
        'citizenship',
        'national_id',
        'national_insurance_number',
        'first_name',
        'last_name',
        'first_address_line',
        'postal_code',
        'phone',
        'title',
        'hourly_rate',
        'start_date',
        'is_pilot'
      ],
      properties: {
        citizenship: {
          bsonType: 'string',
          'enum': [
            'UK',
            'Non-UK'
          ],
          description: 'Citizenship status'
        },
        national_id: {
          bsonType: 'string',
          description: 'National ID'
        },
        national_insurance_number: {
          bsonType: 'string',
          description: 'UK National Insurance Number'
        },
        first_name: {
          bsonType: 'string',
          description: 'Employee\'s first name'
        },
        middle_name: {
          bsonType: 'string',
          description: 'Employee\'s middle name'
        },
        last_name: {
          bsonType: 'string',
          description: 'Employee\'s last name'
        },
        first_address_line: {
          bsonType: 'string',
          description: 'First address line of the employee (Street name and
number)'
        },
        second_address_line: {
          bsonType: 'string',
          description: 'Second address line of the employee (Flat number if
applicable)'
        },
        postal_code: {
          bsonType: 'string',
          description: 'Employee\'s postal code'
        },
        phone: {
          bsonType: 'string',
          description: 'Employee\'s phone number'
        },
        title: {
          bsonType: 'string',
          'enum': [
            'Booking clerk',
            'Maintenance staff',
            'Cabin staff',
            'Pilot'
          ],
          description: 'Employee title'
        },
        position: {
          bsonType: 'string',
          'enum': [
            'Trainee',
            'Grade One Crew',
            'Grade Two Crew',
            'Captain',
            'Second Officer',
            'Relief Pilot',
            'Supervisor',
            'Senior',
            'Junior'
          ],
          description: 'Employee Position'
        },
        hourly_rate: {
          bsonType: 'double',
          minimum: 0,
          description: 'Employee\'s hourly rate'
        },
        start_date: {
          bsonType: 'date',
          description: 'Employment start date'
        },
        is_pilot: {
          bsonType: 'bool',
          description: 'Is the employee a pilot'
        },
        last_fit_for_flight_date: {
          bsonType: 'date',
          description: 'The last fit for flight check-up date'
        }
      }
    }
  }
})

// Create flight collection with data validator on data type and value
db.createCollection("flight", {
  validator: {
    $jsonSchema: {
      bsonType: 'object',
      required: [
        'plane_id',
        'departure_date',
        'arrival_date',
        'departure_airport_id',
        'arrival_airport_id',
        'pilot_id',
        'co_pilot_id',
        'cabin_crew_id_list',
        'total_booked_seat',
        'total_cost'
      ],
      properties: {
        plane_id: {
          bsonType: 'objectId',
          description: 'Plane ID of the flight'
        },
        departure_date: {
          bsonType: 'date',
          description: 'Date and time the flight departs'
        },
        arrival_date: {
          bsonType: 'date',
          description: 'Date and time the flight arrives'
        },
        flight_length: {
          bsonType: 'double',
          description: 'Flight length in hours'
        },
        departure_airport_id: {
          bsonType: 'objectId',
          description: 'Airport ID the flight departs from'
        },
        arrival_airport_id: {
          bsonType: 'objectId',
          description: 'Airport ID the flight arrives'
        },
        pilot_id: {
          bsonType: 'objectId',
          description: 'Employee ID of the pilot'
        },
        co_pilot_id: {
          bsonType: 'objectId',
          description: 'Employee ID of the co-pilot'
        },
        cabin_crew_id_list: {
          bsonType: 'array',
          description: 'Employee ID list of the crew except for pilots'
        },
        total_booked_seat: {
          bsonType: 'int',
          minimum: 0,
          description: 'The total seat number has been booked on the flight'
        },
        total_cost: {
          bsonType: 'double',
          minimum: 0,
          description: 'The total cost to operate the flight'
        }
      }
    }
  }
})

// Create plane collection with data validator on data type and value
db.createCollection("plane", {
  validator: {
    $jsonSchema: {
      bsonType: 'object',
      required: [
        'manufacturer',
        'model',
        'flying_range',
        'service_length',
        'first_use_date',
        'status',
        'seating_capacity'
      ],
      properties: {
        manufacturer: {
          bsonType: 'string',
          description: 'The corporation name that makes the plane'
        },
        model: {
          bsonType: 'string',
          description: 'The plane model'
        },
        flying_range: {
          bsonType: 'int',
          minimum: 0,
          description: 'The flying range of the plane (km)'
        },
        service_length: {
          bsonType: 'int',
          minimum: 1,
          description: 'The plane service length (total flights)'
        },
        first_use_date: {
          bsonType: 'date',
          description: 'The first date the plane being used'
        },
        status: {
          bsonType: 'string',
          enum: ['Working', 'Being repaired', 'Upgrading'],
          description: 'The plane status'
        },
        seating_capacity: {
          bsonType: 'int',
          minimum: 1,
          description: 'The total number of seat of the plane'
        }
      }
    }
  }
})

// Create airport collection with data validator on data type and value
db.createCollection("airport", {
  validator: {
    $jsonSchema: {
      bsonType: 'object',
      required: [
        'airport_name',
        'city',
        'country',
        'hourly_rate',
        'refueling_rate'
      ],
      properties: {
        airport_name: {
          bsonType: 'string',
          description: 'The airport name'
        },
        city: {
          bsonType: 'string',
          description: 'The airport\'s city'
        },
        country: {
          bsonType: 'string',
          description: 'The airport\'s country'
        },
        hourly_rate: {
          bsonType: 'double',
          minimum: 0,
          description: 'The hourly rate of the airport for the time the plane
stops there'
        },
        refueling_rate: {
          bsonType: 'double',
          minimum: 0,
          description: 'The refueling rate of the airport'
        }
      }
    }
  }
})

// Create journey collection with data validator on data type and value
db.createCollection("journey", {
  validator: {
    $jsonSchema: {
      bsonType: 'object',
      required: [
        'number_of_flight',
        'flight_id_list'
      ],
      properties: {
        number_of_flight: {
          bsonType: 'int',
          minimum: 1,
          description: 'The total number of flights in this fourney'
        },
        flight_id_list: {
          bsonType: 'array',
          description: 'List of flight ID the journey has'
        },
        total_cost: {
          bsonType: 'double',
          minimum: 0,
          description: 'The total cost to operate the journey'
        },
        total_revenue: {
          bsonType: 'double',
          minimum: 0,
          description: 'The total revenue of the journey'
        }
      }
    }
  }
})

// Create booking collection with data validator on data type and value
db.createCollection("booking", {
  validator: {
    $jsonSchema: {
      bsonType: 'object',
      required: [
        'journey_id',
        'total_fee'
      ],
      properties: {
        journey_id: {
          bsonType: 'objectId',
          description: 'The journey ID this booking places for'
        },
        total_fee: {
          bsonType: 'double',
          minimum: 0,
          description: 'Total fee the customers pay for the booking'
        }
      }
    }
  }
})
```

1.4 Data population:

This coursework contains the below code in the data_population_code.js script.

```
// Insert data into the employee collection
db.employee.insertMany(
   [{
      _id: ObjectId('61c48554373720d630209560'),
      citizenship: 'UK',
      national_id: 'C2257788',
      national_insurance_number: 'AB123456C',
      first_name: 'Job',
      last_name: 'Steve',
      first_address_line: '1 Meath Crescent',
      postal_code: 'E2 0AQ',
      phone: '+44 7777 444 999',
      title: 'Pilot',
      position: 'Captain',
      hourly_rate: 101,
      start_date: new Date("2018-06-05"),
      is_pilot: true,
      last_fit_for_flight_date: new Date("2019-02-05")
   }, {
      _id: ObjectId('61c48554373720d630209561'),
      citizenship: 'Non-UK',
      national_id: 'V5527788',
      national_insurance_number: 'AB312456C',
      first_name: 'Gate',
      last_name: 'Bill',
      first_address_line: '3 Meath Crescent',
      postal_code: 'E2 7AQ',
      phone: '+44 7777 666 999',
      title: 'Pilot',
      position: 'Trainee',
      hourly_rate: 40,
      start_date: new Date("2021-04-03"),
      is_pilot: true,
      last_fit_for_flight_date: new Date("2019-04-05")
   }, {
      _id: ObjectId('61c48554373720d630209562'),
      citizenship: 'Non-UK',
      national_id: 'L2527788',
      national_insurance_number: 'AB321456C',
      first_name: 'Musk',
      last_name: 'Elon',
      first_address_line: '4 Meath Crescent',
      postal_code: 'E2 6AQ',
      phone: '+44 7777 777 999',
      title: 'Pilot',
      position: 'Second Officer',
      hourly_rate: 60,
      start_date: new Date("2019-03-01"),
      is_pilot: true,
      last_fit_for_flight_date: new Date("2020-03-05")
   }, {
      _id: ObjectId('61c48554373720d630209563'),
      citizenship: 'UK',
      national_id: 'B2759988',
      national_insurance_number: 'AB666444B',
      first_name: 'Buffet',
      last_name: 'Warren',
      first_address_line: '6 Meath Crescent',
      postal_code: 'E2 5AQ',
      phone: '+44 7777 999 999',
      title: 'Pilot',
      position: 'Captain',
      hourly_rate: 111,
      start_date: new Date("2017-06-01"),
      is_pilot: true,
      last_fit_for_flight_date: new Date("2021-08-05")
   }, {
      _id: ObjectId('61c48554373720d630209564'),
      citizenship: 'Non-UK',
      national_id: 'D7759988',
      national_insurance_number: 'AB999222D',
      first_name: 'Kennedy',
      last_name: 'John',
      first_address_line: '7 Meath Crescent',
      postal_code: 'E2 4AQ',
      phone: '+44 2222 999 999',
      title: 'Cabin staff',
      position: 'Grade One Crew',
      hourly_rate: 41,
      start_date: new Date("2011-06-01"),
      is_pilot: false
   }, {
      _id: ObjectId('61c48554373720d630209565'),
      citizenship: 'Non-UK',
      national_id: 'E6759988',
      national_insurance_number: 'AB222111E',
      first_name: 'Putin',
      last_name: 'Vladimir',
      first_address_line: '8 Meath Crescent',
      postal_code: 'E2 1AQ',
      phone: '+44 3333 999 999',
      title: 'Cabin staff',
      position: 'Supervisor',
      hourly_rate: 47,
      start_date: new Date("2012-06-01"),
      is_pilot: false
   }, {
      _id: ObjectId('61c48554373720d630209566'),
      citizenship: 'Non-UK',
      national_id: 'D2259988',
      national_insurance_number: 'AB999333F',
      first_name: 'Kennedy',
      last_name: 'John Jr.',
      first_address_line: '77 Meath Crescent',
      postal_code: 'E2 1AQ',
      phone: '+44 2222 111 999',
      title: 'Cabin staff',
      position: 'Grade One Crew',
      hourly_rate: 42,
      start_date: new Date("2009-08-01"),
      is_pilot: false
   }, {
      _id: ObjectId('61c48554373720d630209567'),
      citizenship: 'Non-UK',
      national_id: 'E1159988',
      national_insurance_number: 'AB222222V',
      first_name: 'Putin',
      last_name: 'Vladimir Jr.',
      first_address_line: '88 Meath Crescent',
      postal_code: 'E2 4AQ',
      phone: '+44 3333 112 999',
      title: 'Cabin staff',
      position: 'Grade Two Crew',
      hourly_rate: 38,
      start_date: new Date("2013-07-01"),
      is_pilot: false
   }, {
      _id: ObjectId('61c48554373720d630209568'),
      citizenship: 'Non-UK',
      national_id: 'F4759988',
      national_insurance_number: 'AB456783F',
      first_name: 'Zedong',
      last_name: 'Mao',
      first_address_line: '9 Meath Crescent',
      postal_code: 'E2 2AQ',
      phone: '+44 8888 999 999',
      title: 'Booking clerk',
      position: 'Supervisor',
      hourly_rate: 36,
      start_date: new Date("2020-06-01"),
      is_pilot: false
   }, {
      _id: ObjectId('61c48554373720d630209569'),
      citizenship: 'Non-UK',
      national_id: 'G3759988',
      national_insurance_number: 'AB984938L',
      first_name: 'John',
      last_name: 'Ken',
      first_address_line: '10 Meath Crescent',
      postal_code: 'E2 3AQ',
      phone: '+44 4444 999 999',
      title: 'Maintenance staff',
      position: 'Senior',
      hourly_rate: 77,
      start_date: new Date("2014-06-01"),
      is_pilot: false
   }]
)

// Insert data into the plane collection
db.plane.insertMany(
   [{
      _id: ObjectId('61c4a2bf373720d63020959a'),
      manufacturer: 'Boeing',
      model: '777',
      flying_range: NumberInt(17205),
      service_length: NumberInt(40000),
      first_use_date: new Date("2010-06-01"),
      status: 'Working',
      seating_capacity: NumberInt(328)
   }, {
      _id: ObjectId('61c4a2bf373720d63020959b'),
      manufacturer: 'Boeing',
      model: '747',
      flying_range: NumberInt(14815),
      service_length: NumberInt(20000),
      first_use_date: new Date("2011-01-06"),
      status: 'Working',
      seating_capacity: NumberInt(467)
   }, {
      _id: ObjectId('61c4a2bf373720d63020959c'),
      manufacturer: 'Airbus',
      model: 'A380',
      flying_range: NumberInt(15700),
      service_length: NumberInt(24000),
      first_use_date: new Date("2009-02-03"),
      status: 'Working',
      seating_capacity: NumberInt(853)
   }, {
      _id: ObjectId('61c4a2bf373720d63020959d'),
      manufacturer: 'Airbus',
      model: 'A380',
      flying_range: NumberInt(15700),
      service_length: NumberInt(24000),
      first_use_date: new Date("2009-11-11"),
      status: 'Upgrading',
      seating_capacity: NumberInt(853)
   }, {
      _id: ObjectId('61c4a2bf373720d63020959e'),
      manufacturer: 'Airbus',
      model: 'A380',
      flying_range: NumberInt(15700),
      service_length: NumberInt(24000),
      first_use_date: new Date("2007-11-11"),
      status: 'Being repaired',
      seating_capacity: NumberInt(853)
   }, {
      _id: ObjectId('61c4a2bf373720d63020959f'),
      manufacturer: 'Boeing',
      model: '707',
      flying_range: NumberInt(10650),
      service_length: NumberInt(20000),
      first_use_date: new Date("2012-01-06"),
      status: 'Working',
      seating_capacity: NumberInt(189)
   }, {
      _id: ObjectId('61c4a2bf373720d63020958a'),
      manufacturer: 'Boeing',
      model: '707',
      flying_range: NumberInt(10650),
      service_length: NumberInt(20000),
      first_use_date: new Date("2002-01-06"),
      status: 'Being repaired',
      seating_capacity: NumberInt(189)
   }])

// Insert data into the airport collection
db.airport.insertMany(
   [{
      _id: ObjectId('61c4a2bf373720d63020951a'),
      airport_name: 'Noi Bai International Airport',
      city: 'Hanoi',
      country: 'Vietnam',
      hourly_rate: 4.25,
      refueling_rate: 1.35
   }, {
      _id: ObjectId('61c4a2bf373720d63020952a'),
      airport_name: 'Heathrow Airport',
      city: 'London',
      country: 'United Kingdom',
      hourly_rate: 6.25,
      refueling_rate: 1.4
   }, {
      _id: ObjectId('61c4a2bf373720d63020953a'),
      airport_name: 'Denver International Airport',
      city: 'Denver',
      country: 'United State',
      hourly_rate: 7.25,
      refueling_rate: 1.3
   }, {
      _id: ObjectId('61c4a2bf373720d63020954a'),
      airport_name: 'Singapore Changi Airport',
      city: 'Singapore',
      country: 'Singapore',
      hourly_rate: 9.25,
      refueling_rate: 1.28
   }, {
      _id: ObjectId('61c4a2bf373720d63020955a'),
      airport_name: 'Los Angeles International Airport',
      city: 'Los Angeles',
      country: 'United State',
      hourly_rate: 7,
      refueling_rate: 1.32
   }, {
      _id: ObjectId('61c4a2bf373720d63020956a'),
      airport_name: 'Hamad International Airport',
      city: 'Doha',
      country: 'Qatar',
      hourly_rate: 9,
      refueling_rate: 1.35
   }])

// Insert data into the flight collection
db.flight.insertMany(
   [{
      _id: ObjectId("61c5df78373720d630209601"),
      plane_id: ObjectId('61c4a2bf373720d63020959f'),
      departure_date: new Date('2018-06-12T10:55:22'),
      arrival_date: new Date('2018-06-13T00:30:25'),
      departure_airport_id: ObjectId('61c4a2bf373720d63020952a'),
      arrival_airport_id: ObjectId('61c4a2bf373720d63020956a'),
      pilot_id: ObjectId('61c48554373720d630209560'),
      co_pilot_id: ObjectId('61c48554373720d630209562'),
      cabin_crew_id_list: [ObjectId('61c48554373720d630209564'),
ObjectId('61c48554373720d630209565'), ObjectId('61c48554373720d630209566')],
      total_booked_seat: NumberInt(153),
      total_cost: 1684
   }, {
      _id: ObjectId("61c5df78373720d630209602"),
      plane_id: ObjectId('61c4a2bf373720d63020959c'),
      departure_date: new Date('2018-06-14T17:10:16'),
      arrival_date: new Date('2018-06-15T04:11:16'),
      departure_airport_id: ObjectId('61c4a2bf373720d63020953a'),
      arrival_airport_id: ObjectId('61c4a2bf373720d63020955a'),
      pilot_id: ObjectId('61c48554373720d630209563'),
      co_pilot_id: ObjectId('61c48554373720d630209562'),
      cabin_crew_id_list: [ObjectId('61c48554373720d630209565'),
ObjectId('61c48554373720d630209567'), ObjectId('61c48554373720d630209566')],
      total_booked_seat: NumberInt(678),
      total_cost: 1692
   }, {
      _id: ObjectId("61c5df78373720d630209603"),
      plane_id: ObjectId('61c4a2bf373720d63020959c'),
      departure_date: new Date('2018-06-17T16:00:11'),
      arrival_date: new Date('2018-06-18T05:41:56'),
      departure_airport_id: ObjectId('61c4a2bf373720d63020954a'),
      arrival_airport_id: ObjectId('61c4a2bf373720d63020955a'),
      pilot_id: ObjectId('61c48554373720d630209563'),
      co_pilot_id: ObjectId('61c48554373720d630209560'),
      cabin_crew_id_list: [ObjectId('61c48554373720d630209566'),
ObjectId('61c48554373720d630209565')],
      total_booked_seat: NumberInt(853),
      total_cost: 1462
   }, {
      _id: ObjectId("61c5df78373720d630209604"),
      plane_id: ObjectId('61c4a2bf373720d63020959b'),
      departure_date: new Date('2018-06-19T19:26:39'),
      arrival_date: new Date('2018-06-20T05:41:58'),
      departure_airport_id: ObjectId('61c4a2bf373720d63020951a'),
      arrival_airport_id: ObjectId('61c4a2bf373720d63020956a'),
      pilot_id: ObjectId('61c48554373720d630209560'),
      co_pilot_id: ObjectId('61c48554373720d630209561'),
      cabin_crew_id_list: [ObjectId('61c48554373720d630209564'),
ObjectId('61c48554373720d630209566'), ObjectId('61c48554373720d630209567')],
      total_booked_seat: NumberInt(456),
      total_cost: 1734.5
   }, {
      _id: ObjectId("61c5df78373720d630209605"),
      plane_id: ObjectId('61c4a2bf373720d63020959c'),
      departure_date: new Date('2018-06-22T03:40:02'),
      arrival_date: new Date('2018-06-22T18:21:21'),
      departure_airport_id: ObjectId('61c4a2bf373720d63020955a'),
      arrival_airport_id: ObjectId('61c4a2bf373720d63020954a'),
      pilot_id: ObjectId('61c48554373720d630209560'),
      co_pilot_id: ObjectId('61c48554373720d630209563'),
      cabin_crew_id_list: [ObjectId('61c48554373720d630209567'),
ObjectId('61c48554373720d630209565'), ObjectId('61c48554373720d630209566')],
      total_booked_seat: NumberInt(645),
      total_cost: 1794
   }, {
      _id: ObjectId("61c5df78373720d630209606"),
      plane_id: ObjectId('61c4a2bf373720d63020959c'),
      departure_date: new Date('2018-06-24T20:37:18'),
      arrival_date: new Date('2018-06-25T20:08:20'),
      departure_airport_id: ObjectId('61c4a2bf373720d63020953a'),
      arrival_airport_id: ObjectId('61c4a2bf373720d63020954a'),
      pilot_id: ObjectId('61c48554373720d630209563'),
      co_pilot_id: ObjectId('61c48554373720d630209561'),
      cabin_crew_id_list: [ObjectId('61c48554373720d630209566'),
ObjectId('61c48554373720d630209567')],
      total_booked_seat: NumberInt(785),
      total_cost: 1534.7
   }, {
      _id: ObjectId("61c5df78373720d630209607"),
      plane_id: ObjectId('61c4a2bf373720d63020959f'),
      departure_date: new Date('2018-06-27T03:41:27'),
      arrival_date: new Date('2018-06-28T04:27:18'),
      departure_airport_id: ObjectId('61c4a2bf373720d63020951a'),
      arrival_airport_id: ObjectId('61c4a2bf373720d63020954a'),
      pilot_id: ObjectId('61c48554373720d630209563'),
      co_pilot_id: ObjectId('61c48554373720d630209562'),
      cabin_crew_id_list: [ObjectId('61c48554373720d630209565'),
ObjectId('61c48554373720d630209566')],
      total_booked_seat: NumberInt(189),
      total_cost: 1646.4
   }, {
      _id: ObjectId("61c5df78373720d630209608"),
      plane_id: ObjectId('61c4a2bf373720d63020959f'),
      departure_date: new Date('2018-06-30T05:44:36'),
      arrival_date: new Date('2018-06-30T16:58:23'),
      departure_airport_id: ObjectId('61c4a2bf373720d63020956a'),
      arrival_airport_id: ObjectId('61c4a2bf373720d63020951a'),
      pilot_id: ObjectId('61c48554373720d630209563'),
      co_pilot_id: ObjectId('61c48554373720d630209562'),
      cabin_crew_id_list: [ObjectId('61c48554373720d630209565'),
ObjectId('61c48554373720d630209567')],
      total_booked_seat: NumberInt(189),
      total_cost: 1546
   }, {
      _id: ObjectId("61c5df78373720d630209609"),
      plane_id: ObjectId('61c4a2bf373720d63020959c'),
      departure_date: new Date('2018-07-03T00:48:00'),
      arrival_date: new Date('2018-07-03T12:26:25'),
      departure_airport_id: ObjectId('61c4a2bf373720d63020951a'),
      arrival_airport_id: ObjectId('61c4a2bf373720d63020954a'),
      pilot_id: ObjectId('61c48554373720d630209562'),
      co_pilot_id: ObjectId('61c48554373720d630209563'),
      cabin_crew_id_list: [ObjectId('61c48554373720d630209567'),
ObjectId('61c48554373720d630209564'), ObjectId('61c48554373720d630209565')],
      total_booked_seat: NumberInt(234),
      total_cost: 1764
   }, {
      _id: ObjectId("61c5df78373720d63020960a"),
      plane_id: ObjectId('61c4a2bf373720d63020959a'),
      departure_date: new Date('2018-07-05T16:37:23'),
      arrival_date: new Date('2018-07-06T04:23:28'),
      departure_airport_id: ObjectId('61c4a2bf373720d63020956a'),
      arrival_airport_id: ObjectId('61c4a2bf373720d63020951a'),
      pilot_id: ObjectId('61c48554373720d630209563'),
      co_pilot_id: ObjectId('61c48554373720d630209561'),
      cabin_crew_id_list: [ObjectId('61c48554373720d630209567'),
ObjectId('61c48554373720d630209564')],
      total_booked_seat: NumberInt(300),
      total_cost: 1744.5
   }, {
      _id: ObjectId("61c5df78373720d63020960b"),
      plane_id: ObjectId('61c4a2bf373720d63020959f'),
      departure_date: new Date('2018-07-07T20:25:43'),
      arrival_date: new Date('2018-07-08T08:25:27'),
      departure_airport_id: ObjectId('61c4a2bf373720d63020951a'),
      arrival_airport_id: ObjectId('61c4a2bf373720d63020955a'),
      pilot_id: ObjectId('61c48554373720d630209563'),
      co_pilot_id: ObjectId('61c48554373720d630209561'),
      cabin_crew_id_list: [ObjectId('61c48554373720d630209564'),
ObjectId('61c48554373720d630209566'), ObjectId('61c48554373720d630209565')],
      total_booked_seat: NumberInt(180),
      total_cost: 1666.5
   }])
// Update flight_length field for all the document of flight collection
db.flight.update({}, [{
   $set: {
      flight_length: {
         $round: [{
            $divide: [{
               $subtract: ["$arrival_date", "$departure_date"]
            }, 60 * 60000]
         }, 3]
      }
   }
}], {
   "multi": true
})

// Insert data into journey collection, using ID from the flight collection
db.journey.insertMany(
   [{
      _id: ObjectId("61c5dfff373720d63020960c"),
      number_of_flight: NumberInt(1),
      flight_id_list: [ObjectId("61c5df78373720d630209601")],
      total_cost: 1784,
      total_revenue: 2784
   }, {
      _id: ObjectId("61c5dfff373720d63020960d"),
      number_of_flight: NumberInt(2),
      flight_id_list: [ObjectId("61c5df78373720d630209602"),
ObjectId("61c5df78373720d630209605")],
      total_cost: 3686,
      total_revenue: 5686
   }, {
      _id: ObjectId("61c5dfff373720d63020960e"),
      number_of_flight: NumberInt(1),
      flight_id_list: [ObjectId("61c5df78373720d630209603")],
      total_cost: 1690,
      total_revenue: 2690
   }, {
      _id: ObjectId("61c5dfff373720d63020960f"),
      number_of_flight: NumberInt(1),
      flight_id_list: [ObjectId("61c5df78373720d630209604")],
      total_cost: 1834.5,
      total_revenue: 4834.5
   }, {
      _id: ObjectId("61c5dfff373720d630209610"),
      number_of_flight: NumberInt(1),
      flight_id_list: [ObjectId("61c5df78373720d630209606")],
      total_cost: 1834.7,
      total_revenue: 5834.7
   }, {
      _id: ObjectId("61c5dfff373720d630209611"),
      number_of_flight: NumberInt(1),
      flight_id_list: [ObjectId("61c5df78373720d630209607")],
      total_cost: 1746.4,
      total_revenue: 6746.4
   }, {
      _id: ObjectId("61c5dfff373720d630209612"),
      number_of_flight: NumberInt(1),
      flight_id_list: [ObjectId("61c5df78373720d630209608")],
      total_cost: 1746,
      total_revenue: 4746
   }, {
      _id: ObjectId("61c5dfff373720d630209613"),
      number_of_flight: NumberInt(1),
      flight_id_list: [ObjectId("61c5df78373720d630209609")],
      total_cost: 1864,
      total_revenue: 3864
   }, {
      _id: ObjectId("61c5dfff373720d630209614"),
      number_of_flight: NumberInt(1),
      flight_id_list: [ObjectId("61c5df78373720d63020960a")],
      total_cost: 1944.5,
      total_revenue: 2944
   }, {
      _id: ObjectId("61c5dfff373720d630209615"),
      number_of_flight: NumberInt(1),
      flight_id_list: [ObjectId("61c5df78373720d63020960b")],
      total_cost: 1766.5,
      total_revenue: 3766
   }])

// Insert data into booking collection, using ID from the journey collection
db.booking.insertMany(
   [{
      journey_id: ObjectId("61c5dfff373720d63020960c"),
      total_fee: 3070
   }, {
      journey_id: ObjectId("61c5dfff373720d63020960d"),
      total_fee: 5892
```

```
  }, {
    journey_id: ObjectId("61c5dfff373720d63020960e"),
    total_fee: 2890
  }, {
    journey_id: ObjectId("61c5dfff373720d63020960f"),
    total_fee: 4890
  }, {
    journey_id: ObjectId("61c5dfff373720d630209610"),
    total_fee: 6040
  }, {
    journey_id: ObjectId("61c5dfff373720d630209611"),
    total_fee: 3450
  }, {
    journey_id: ObjectId("61c5dfff373720d630209612"),
    total_fee: 5430
  }, {
    journey_id: ObjectId("61c5dfff373720d630209613"),
    total_fee: 8570
  }, {
    journey_id: ObjectId("61c5dfff373720d63020960c"),
    total_fee: 4730
  }, {
    journey_id: ObjectId("61c5dfff373720d63020960d"),
    total_fee: 2420
  }, {
    journey_id: ObjectId("61c5dfff373720d63020960e"),
    total_fee: 2900
  }, {
    journey_id: ObjectId("61c5dfff373720d63020960f"),
    total_fee: 4900
  }, {
    journey_id: ObjectId("61c5dfff373720d630209610"),
    total_fee: 6000
  }, {
    journey_id: ObjectId("61c5dfff373720d630209611"),
    total_fee: 3500
  }, {
    journey_id: ObjectId("61c5dfff373720d630209612"),
    total_fee: 5450
  }, {
    journey_id: ObjectId("61c5dfff373720d630209613"),
    total_fee: 8600
  }, {
    journey_id: ObjectId("61c5dfff373720d630209614"),
    total_fee: 4700
```

```
   }, {
      journey_id: ObjectId("61c5dfff373720d630209615"),
      total_fee: 2450
   }])
```

1.5 Set of 12 queries:

1. First of all, when exploring a new system, a check-up for to database status would be necessary.
   This query demonstrates the usage of the stats function in MongoDB.

```
> db.stats()

Expected result:
{
        "db" : "test",
        "collections" : 7,
        "views" : 0,
        "objects" : 62,
        "avgObjSize" : 190.1451612903226,
        "dataSize" : 11789,
        "storageSize" : 225280,
        "numExtents" : 0,
        "indexes" : 7,
        "indexSize" : 225280,
        "scaleFactor" : 1,
        "fsUsedSize" : 107585744896,
        "fsTotalSize" : 528209670144,
        "ok" : 1
}
```

2. After viewing the overview stats of the system, there is an abnormality in its data model. Since the data model diagram only has 6 collections, however, these stats indicate there are 7 collections in the database. Therefore, a drill-down check-up for the list of the collection is needed.
   This query demonstrates the usage of the find function in MongoDB.

```
> db.getCollectionInfos()

Expected result: There is a blank collection named “test” aside from the
expected six collections in the data model.
The result includes all the collections specifications, validation rules,
and descriptions. The actual output of this query is in the queries.js file.
```

3. After addressing the abnormality of the system, a brief check for total documents in each collections is expected to assure its data completeness.
   This query demonstrates the usage of the count function in MongoDB.

```
> db.flight.count()

Expected result:
11
```

4. Find out the list of pilots with captain positions.
   This query demonstrates the usage of the find function in MongoDB.

```
> db.employee.find({
    "position": {
        "$eq": "Captain"
    }
})

Expected result:
{
    "_id": ObjectId("61c48554373720d630209560"),
    "citizenship": "UK",
    "national_id": "C2257788",
    "national_insurance_number": "AB123456C",
    "first_name": "Job",
    "last_name": "Steve",
    "first_address_line": "1 Meath Crescent",
    "postal_code": "E2 0AQ",
    "phone": "+44 7777 444 999",
    "title": "Pilot",
    "position": "Captain",
    "hourly_rate": 101,
    "start_date": ISODate("2018-06-05T00:00:00Z"),
    "is_pilot": true,
    "last_fit_for_flight_date": ISODate("2019-02-05T00:00:00Z")
} {
    "_id": ObjectId("61c48554373720d630209563"),
    "citizenship": "UK",
    "national_id": "B2759988",
    "national_insurance_number": "AB666444B",
    "first_name": "Buffet",
    "last_name": "Warren",
    "first_address_line": "6 Meath Crescent",
    "postal_code": "E2 5AQ",
    "phone": "+44 7777 999 999",
    "title": "Pilot",
    "position": "Captain",
    "hourly_rate": 111,
    "start_date": ISODate("2017-06-01T00:00:00Z"),
    "is_pilot": true,
    "last_fit_for_flight_date": ISODate("2021-08-05T00:00:00Z")
}
```

5. Find out which airport has the highest hourly rate for stopping airplanes in transit.
   This query demonstrates the usage of the sort and limit function in MongoDB.

```
> db.airport.find({}).sort({
    hourly_rate: -1
}).limit(1)

Expected result:
{
    "_id": ObjectId("61c4a2bf373720d63020954a"),
    "airport_name": "Singapore Changi Airport",
    "city": "Singapore",
    "country": "Singapore",
    "hourly_rate": 9.25,
    "refueling_rate": 1.28
}
```

6. Assessing the company current operating status of all the plane .
   This query demonstrates the usage of a simple aggregation framework using $group and $sort in MongoDB.

```
> db.plane.aggregate([{
    "$group": {
        _id: "$status",
        count: {
            $sum: 1
        }
    }
}, {
    "$sort": {
        count: -1
    }
}])

Expected result:
{
    "_id": "Working",
    "count": 4
} {
    "_id": "Being repaired",
    "count": 2
} {
    "_id": "Upgrading",
    "count": 1
}
```

7. Find out which most frequent flight. This might be useful if the company needs to maximize profit.
   This query demonstrates the usage of a simple aggregation framework using $group, $sum, $sort, and $limit in MongoDB.

```
> db.flight.aggregate([{
    "$group": {
        _id: {
            dep: "$departure_airport_id",
            arr: "$arrival_airport_id"
        },
        count: {
            "$sum": 1
        }
    }
}, {
    "$sort": {
        count: -1
    }
}, {
    "$limit": 1
}])

Expected result:
{
    "_id": {
        "dep": ObjectId("61c4a2bf373720d63020956a"),
        "arr": ObjectId("61c4a2bf373720d63020951a")
    },
    "count": 2
}
```

8. For the same business use case above, the code below optimizes the interpretability of the result. Instead of showing objectId of the departure and arrival airport, a look-up between collections will be conducted to get their name.
   This query demonstrates the usage of a simple aggregation framework using $lookup and pretty function in MongoDB.

```
> db.flight.aggregate([{
    "$lookup": {
        from: "airport",
        localField: "departure_airport_id",
        foreignField: "_id",
        as: "dep"
    }
}, {
    "$lookup": {
        from: "airport",
        localField: "arrival_airport_id",
        foreignField: "_id",
        as: "arr"
    }
}, {
    "$group": {
        _id: {
            departure_airport: "$dep.airport_name",
            arrival_airport: "$arr.airport_name"
        },
        count: {
            "$sum": 1
        }
    }
}, {
    "$sort": {
        count: -1
    }
}, {
    "$limit": 1
}]).pretty()

Expected result:
{
    "_id" : {
            "departure_airport" : [
                    "Hamad International Airport"
            ],
            "arrival_airport" : [
                    "Noi Bai International Airport"
            ]
    },
    "count" : 2
}
```

9. Assessing flight frequency per day of the week. This insight could be good for the resource allocation plan of the company.
   This query demonstrates the usage of the $dayOfWeek and $sum function in MongoDB.

```
db.flight.aggregate(
    [{
        "$project": {
            _id: 0,
            day_of_week_uk: {
                "$dayOfWeek": {
                    date: "$departure_date",
                    timezone: "Europe/London"
                }
            }
        }
    }, {
        "$group": {
            _id: {
                day_of_week_uk: "$day_of_week_uk"
            },
            count: {
                "$sum": 1
            }
        }
    }, {
        "$sort": {
            count: -1
        }
    }]
)
```

Expected result:

```
{
    "_id": {
        "day_of_week_uk": 3
    },
    "count": 3
} {
    "_id": {
        "day_of_week_uk": 7
    },
    "count": 2
} {
    "_id": {
        "day_of_week_uk": 5
    },
    "count": 2
} {
    "_id": {
        "day_of_week_uk": 1
    },
    "count": 2
} {
    "_id": {
        "day_of_week_uk": 6
    },
    "count": 1
} {
    "_id": {
        "day_of_week_uk": 4
    },
    "count": 1
}
```

10. Considering the output from query 5, the result is hard to interpret for the user since the day of week is encoded.
    In this query, the $switch will be used for easier interpretation of the result.

```
db.flight.aggregate(
    [{
        "$project": {
            _id: 0,
            day_of_week_uk: {
                "$dayOfWeek": {
                    date: "$departure_date",
                    timezone: "Europe/London"
                }
            }
        }
    }, {
        "$group": {
            _id: {
                day_of_week_uk: {
                    "$switch": {
                        branches: [{
                            case: {
                                "$eq": ["$day_of_week_uk", 1]
                            },
                            then: "Sunday"
                        }, {
                            case: {
                                "$eq": ["$day_of_week_uk", 2]
                            },
                            then: "Monday"
                        }, {
                            case: {
                                "$eq": ["$day_of_week_uk", 3]
                            },
                            then: "Tuesday"
                        }, {
                            case: {
                                "$eq": ["$day_of_week_uk", 4]
                            },
                            then: "Wednesday"
                        }, {
                            case: {
                                "$eq": ["$day_of_week_uk", 5]
                            },
                            then: "Thursday"
                        }, {
                            case: {
                                "$eq": ["$day_of_week_uk", 6]
                            },
                            then: "Friday"
                        }, {
                            case: {
                                "$eq": ["$day_of_week_uk", 7]
                            },
                            then: "Saturday"
                        }, ]
                    }
                }
            },
            count: {
                "$sum": 1
            }
        }
    }, {
        "$sort": {
            count: -1
        }
    }]
)
```

Expected result:

```
{
    "_id": {
        "day_of_week_uk": "Tuesday"
    },
    "count": 3
} {
    "_id": {
        "day_of_week_uk": "Sunday"
    },
    "count": 2
} {
    "_id": {
        "day_of_week_uk": "Thursday"
    },
    "count": 2
} {
    "_id": {
        "day_of_week_uk": "Saturday"
    },
    "count": 2
} {
    "_id": {
        "day_of_week_uk": "Friday"
    },
    "count": 1
} {
    "_id": {
        "day_of_week_uk": "Wednesday"
    },
    "count": 1
}
```

11. For flight planning, the company needs to assess the top three flights destination that has the highest booking rate (in term of the total booked seat versus the plane available seat – this will be called utilization_rate).
    This query demonstrates the usage of a simple aggregation framework using $round and $project function to enhance the interpretability of the output document in MongoDB.

```
> db.flight.aggregate([{
    "$lookup": {
        from: "airport",
        localField: "departure_airport_id",
        foreignField: "_id",
        as: "dep"
    }
}, {
    "$lookup": {
        from: "airport",
        localField: "arrival_airport_id",
        foreignField: "_id",
        as: "arr"
    }
}, {
    "$lookup": {
        from: "plane",
        localField: "plane_id",
        foreignField: "_id",
        as: "plane"
    }
}, {
    "$project": {
        departure_airport: "$dep.airport_name",
        arrival_airport: "$arr.airport_name",
        utilization_rate: {
            "$divide": ["$total_booked_seat", {

            }]
        }
    }
}, {
    "$group": {
        _id: {
            departure_airport: "$departure_airport",
            arrival_airport: "$arrival_airport"
        },
        avg_utilization_rate: {
            "$avg": "$utilization_rate"
        }
    }
}, {
    "$project": {
        departure_airport: "$departure_airport",
        arrival_airport: "$arrival_airport",
        avg_utilization_rate: {
            "$round": ["$avg_utilization_rate", 3]
        }
    }
}, {
    "$sort": {
        avg_utilization_rate: -1
    }
}, {
    "$limit": 3
}]).pretty()
```

Expected result:

```
{
    "_id" : {
            "departure_airport" : [
                    "Singapore Changi Airport"
            ],
            "arrival_airport" : [
                    "Los Angeles International Airport"
            ]
    },
    "avg_utilization_rate" : 1
}
{
    "_id" : {
            "departure_airport" : [
                    "Noi Bai International Airport"
            ],
            "arrival_airport" : [
                    "Hamad International Airport"
            ]
    },
    "avg_utilization_rate" : 0.976
}
{
    "_id" : {
            "departure_airport" : [
                    "Hamad International Airport"
            ],
            "arrival_airport" : [
                    "Noi Bai International Airport"
            ]
    },
    "avg_utilization_rate" : 0.957
}
```

12. For the operationalize process of this system, there will be a need of a aggregation pipeline to calculate total cost of a flight. Below is the code to calculate total cost of a flight by hourly rate of all pilot, co-pilot and the whole cabin crew (attendant).
    This query demonstrates the usage of a simple aggregation framework $unwind for array field in MongoDB.

```
db.flight.aggregate([{
    "$unwind": "$cabin_crew_id_list"
}, {
    "$lookup": {
        from: "employee",
        localField: "cabin_crew_id_list",
        foreignField: "_id",
        as: "emp"
    }
}, {
    "$lookup": {
        from: "employee",
        localField: "pilot_id",
        foreignField: "_id",
        as: "pil"
    }
```

```
}, {
    "$lookup": {
        from: "employee",
        localField: "co_pilot_id",
        foreignField: "_id",
        as: "cop"
    }
}, {
    "$group": {
        _id: "$_id",
        // In this group function, we will use the max function for the
hourly rate of pilot, co-pilot, and flight_length since this number is the
same throughout unwinded documents
        flight_length: {
            "$max": "$flight_length"
        },
        pilot_rate: {
            "$max": {
                "$arrayElemAt": ["$pil.hourly_rate", 0]
            }
        },
        cop_rate: {
            "$max": {
                "$max": {
                    "$arrayElemAt": ["$cop.hourly_rate", 0]
                }
            }
        },
        attendant_rate: {
            "$sum": {
                "$arrayElemAt": ["$emp.hourly_rate", 0]
            }
        }
    }
}, {
    "$project": {
        _id: "$_id",
        total_cost: {
            "$round": [{
                "$multiply": [{
                    "$sum": ["$pilot_rate", "$cop_rate", "$attendant_rate"]
                }, "$flight_length"]
            }, 3]
        }
    }
```

```
}])

Expected result:
{
    "_id": ObjectId("61c5df78373720d630209602"),
    "total_cost": 3283.066
} {
    "_id": ObjectId("61c5df78373720d630209604"),
    "total_cost": 2686.81
} {
    "_id": ObjectId("61c5df78373720d630209601"),
    "total_cost": 3952.944
} {
    "_id": ObjectId("61c5df78373720d63020960a"),
    "total_cost": 2706.64
} {
    "_id": ObjectId("61c5df78373720d63020960b"),
    "total_cost": 3370.876
} {
    "_id": ObjectId("61c5df78373720d630209607"),
    "total_cost": 6438.64
} {
    "_id": ObjectId("61c5df78373720d630209606"),
    "total_cost": 5432.427
} {
    "_id": ObjectId("61c5df78373720d630209605"),
    "total_cost": 4979.571
} {
    "_id": ObjectId("61c5df78373720d630209603"),
    "total_cost": 4122.496
} {
    "_id": ObjectId("61c5df78373720d630209609"),
    "total_cost": 3457.08
} {
    "_id": ObjectId("61c5df78373720d630209608"),
    "total_cost": 2874.88
}
```

Part 2: Performance tuning

2.1 Indexes usage with Explain function

Queries number 10 and 12 will be examined the execution statistics by explaining function with and without indexes. For full explain query and result, please access performance_tuning.js in this coursework folder.

Queries 10: As can be seen, the execution statistics after indexed is significantly faster that the execution time is not reach 1 millisecond comparing to 10 milliseconds before indexing. This method proves to be effective for MongoDB performance tuning.

```
// Using Execution Stats to assess the query
db.flight.explain("executionStats").aggregate([*aggregation pipeline of query
10*])

Before indexes:
"executionStats": {
        "executionSuccess": true,
        "nReturned": 11,
        "executionTimeMillis": 10,
        "totalKeysExamined": 0,
        "totalDocsExamined": 11,
        "executionStages": {
                "stage": "COLLSCAN",
                "nReturned": 11,
                "executionTimeMillisEstimate": 0,
                "works": 13,
                "advanced": 11,
                "needTime": 1,
                "needYield": 0,
                "saveState": 1,
                "restoreState": 1,
                "isEOF": 1,
                "direction": "forward",
                "docsExamined": 11
        }

Add index:
> db.flight.createIndex({plane_id: 1})

{
        "createdCollectionAutomatically": false,
        "numIndexesBefore": 1,
        "numIndexesAfter": 2,
        "ok": 1
}

After indexes:
"executionStats": {
        "executionSuccess": true,
        "nReturned": 11,
        "executionTimeMillis": 0,
        "totalKeysExamined": 0,
        "totalDocsExamined": 11,
        "executionStages": {
                "stage": "COLLSCAN",
                "nReturned": 11,
                "executionTimeMillisEstimate": 0,
                "works": 13,
                "advanced": 11,
                "needTime": 1,
                "needYield": 0,
                "saveState": 1,
                "restoreState": 1,
                "isEOF": 1,
                "direction": "forward",
                "docsExamined": 11
        }
```

Query 12: As can be seen, the execution statistics after indexed is 2 milliseconds faster than before indexed. Even when the dummy data is small, we could see a 20% improvement in our query time. The indexing does not improve the query time significantly like query 10 is because query 12 accesses data from employee collection too. Therefore, if all the collection in this data model is indexed, a huge performance uplift is expected.

```
// Using Execution Stats to assess the query
db.flight.explain("executionStats").aggregate([*aggregation pipeline of query
12*])

Before indexes:
"executionStats": {
        "executionSuccess": true,
        "nReturned": 11,
        "executionTimeMillis": 10,
        "totalKeysExamined": 0,
        "totalDocsExamined": 11,
        "executionStages": {
                "stage": "COLLSCAN",
                "nReturned": 11,
                "executionTimeMillisEstimate": 0,
                "works": 13,
                "advanced": 11,
                "needTime": 1,
                "needYield": 0,
                "saveState": 1,
                "restoreState": 1,
                "isEOF": 1,
                "direction": "forward",
                "docsExamined": 11
        }
```

```
Add index:
> db.flight.createIndex({plane_id: 1})

{
        "createdCollectionAutomatically": false,
        "numIndexesBefore": 1,
        "numIndexesAfter": 2,
        "ok": 1
}

After indexes:
"executionStats": {
        "executionSuccess": true,
        "nReturned": 11,
        "executionTimeMillis": 8,
        "totalKeysExamined": 0,
        "totalDocsExamined": 11,
        "executionStages": {
                "stage": "COLLSCAN",
                "nReturned": 11,
                "executionTimeMillisEstimate": 0,
                "works": 13,
                "advanced": 11,
                "needTime": 1,
                "needYield": 0,
                "saveState": 1,
                "restoreState": 1,
                "isEOF": 1,
                "direction": "forward",
                "docsExamined": 11
        }
```

2.2 MongoDB profiler

There are several command from count, $group, and $lookup that will be examined in the profiler result. For full explain query and result, please access profiler_output.js in this coursework folder.

As can be seen, there are two types of planSummary, RECORD_STORE_FAST_COUNT, and COLLSCAN. As the plan summary RECORD_STORE_FAST_COUNT, we expected the query to be fast but depending on the query, the result could be approximated instead of the exact number since this plan returns results from the database metadata.

As the plan summary COLLSCAN, a collection scan is expected in this query. Therefore, the more data the database has, the longer the queries execution time is expected.

The performance of the below queries could be assessed through millis field returned in the profiler output.

```
// Assessing the profiler output for some of the demonstrated queries
> db.system.profile.find({"op" : {$eq:"command"}} ).pretty()

{
        "op": "command",
        "ns": "test.flight",
        "command": {
                "count": "flight",
                "query": {

                },
                "lsid": {
                        "id": UUID("b0987057-c02c-457d-bed9-0a3e3b08b4b2")
                },
                "$db": "test"
        },
        "keysExamined": 0,
        "docsExamined": 0,
        "numYield": 0,
        "locks": {
                "ReplicationStateTransition": {
                        "acquireCount": {
                                "w": NumberLong(1)
                        }
                },
                "Global": {
                        "acquireCount": {
                                "r": NumberLong(1)
                        }
                },
                "Database": {
                        "acquireCount": {
                                "r": NumberLong(1)
                        }
                },
                "Collection": {
                        "acquireCount": {
                                "r": NumberLong(1)
                        }
                },
                "Mutex": {
                        "acquireCount": {
                                "r": NumberLong(1)
                        }
                }
        },
        "flowControl": {

        },
        "responseLength": 45,
        "protocol": "op_msg",
        "millis": 0,
        "planSummary": "RECORD_STORE_FAST_COUNT",
        "execStats": {
                "stage": "RECORD_STORE_FAST_COUNT",
                "nReturned": 0,
                "executionTimeMillisEstimate": 0,
                "works": 1,
                "advanced": 0,
                "needTime": 0,
                "needYield": 0,
                "saveState": 0,
                "restoreState": 0,
                "isEOF": 1,
                "nCounted": 11,
                "nSkipped": 0
        },
        "ts": ISODate("2021-12-25T16:39:18.171Z"),
        "client": "127.0.0.1",
        "appName": "MongoDB Shell",
        "allUsers": [],
        "user": ""
}

{
        "op": "command",
        "ns": "test.flight",
        "command": {
                "aggregate": "flight",
                "pipeline": [{
                        "$group": {
                                "_id": {
                                        "dep": "$departure_airport_id",
                                        "arr": "$arrival_airport_id"
                                },
                                "count": {
                                        "$sum": 1
                                }
                        }
                }, {
                        "$sort": {
                                "count": -1
                        }
                }, {
                        "$limit": 1
                }],
                "cursor": {

                },
                "lsid": {
                        "id": UUID("b0987057-c02c-457d-bed9-0a3e3b08b4b2")
                },
                "$db": "test"
        },
        "keysExamined": 0,
        "docsExamined": 11,
        "hasSortStage": true,
        "cursorExhausted": true,
        "numYield": 0,
        "nreturned": 1,
        "locks": {
                "ReplicationStateTransition": {
                        "acquireCount": {
                                "w": NumberLong(2)
                        }
                },
                "Global": {
                        "acquireCount": {
                                "r": NumberLong(2)
                        }
                },
                "Database": {
                        "acquireCount": {
                                "r": NumberLong(2)
                        }
                },
                "Collection": {
                        "acquireCount": {
                                "r": NumberLong(2)
                        }
                },
                "Mutex": {
                        "acquireCount": {
                                "r": NumberLong(2)
                        }
                }
        },
        "flowControl": {

        },
        "responseLength": 167,
        "protocol": "op_msg",
        "millis": 0,
        "planSummary": "COLLSCAN",
        "ts": ISODate("2021-12-25T16:39:39.805Z"),
        "client": "127.0.0.1",
        "appName": "MongoDB Shell",
        "allUsers": [],
        "user": ""
}

{
        "op": "command",
        "ns": "test.flight",
        "command": {
                "aggregate": "flight",
                "pipeline": [{
                        "$lookup": {
                                "from": "airport",
                                "localField": "departure_airport_id",
                                "foreignField": "_id",
                                "as": "dep"
                        }
                }, {
                        "$lookup": {
                                "from": "airport",
                                "localField": "arrival_airport_id",
                                "foreignField": "_id",
                                "as": "arr"
                        }
                }, {
                        "$group": {
                                "_id": {
                                        "departure_airport":
"$dep.airport_name",
                                        "arrival_airport": "$arr.airport_name"
                                },
                                "count": {
                                        "$sum": 1
                                }
                        }
                }, {
                        "$sort": {
                                "count": -1
                        }
                }, {
                        "$limit": 1
                }],
                "cursor": {

                },
                "lsid": {
                        "id": UUID("b0987057-c02c-457d-bed9-0a3e3b08b4b2")
                },
                "$db": "test"
        },
        "keysExamined": 0,
        "docsExamined": 11,
        "hasSortStage": true,
        "cursorExhausted": true,
        "numYield": 0,
        "nreturned": 1,
        "queryHash": "A300CFDE",
        "planCacheKey": "A2B33459",
        "locks": {
                "ReplicationStateTransition": {
                        "acquireCount": {
                                "w": NumberLong(47)
                        }
                },
                "Global": {
                        "acquireCount": {
                                "r": NumberLong(47)
                        }
                },
                "Database": {
                        "acquireCount": {
                                "r": NumberLong(47)
                        }
                },
                "Collection": {
                        "acquireCount": {
                                "r": NumberLong(48)
                        }
                },
                "Mutex": {
                        "acquireCount": {
                                "r": NumberLong(47)
                        }
                }
        },
        "flowControl": {

        },
        "responseLength": 251,
        "protocol": "op_msg",
        "millis": 5,
        "planSummary": "COLLSCAN",
        "ts": ISODate("2021-12-25T16:39:48.767Z"),
        "client": "127.0.0.1",
        "appName": "MongoDB Shell",
        "allUsers": [],
        "user": ""
}
```