Queen Mary University of London

ECS650U Semi-Structured Data and Advanced Data Modelling
Coursework 1: MongoDB design and implementation

CW1_UG Group 7:

Mukhtar Ismail
Mikhdad Miah
Cruz Felix, Frank Erasmo

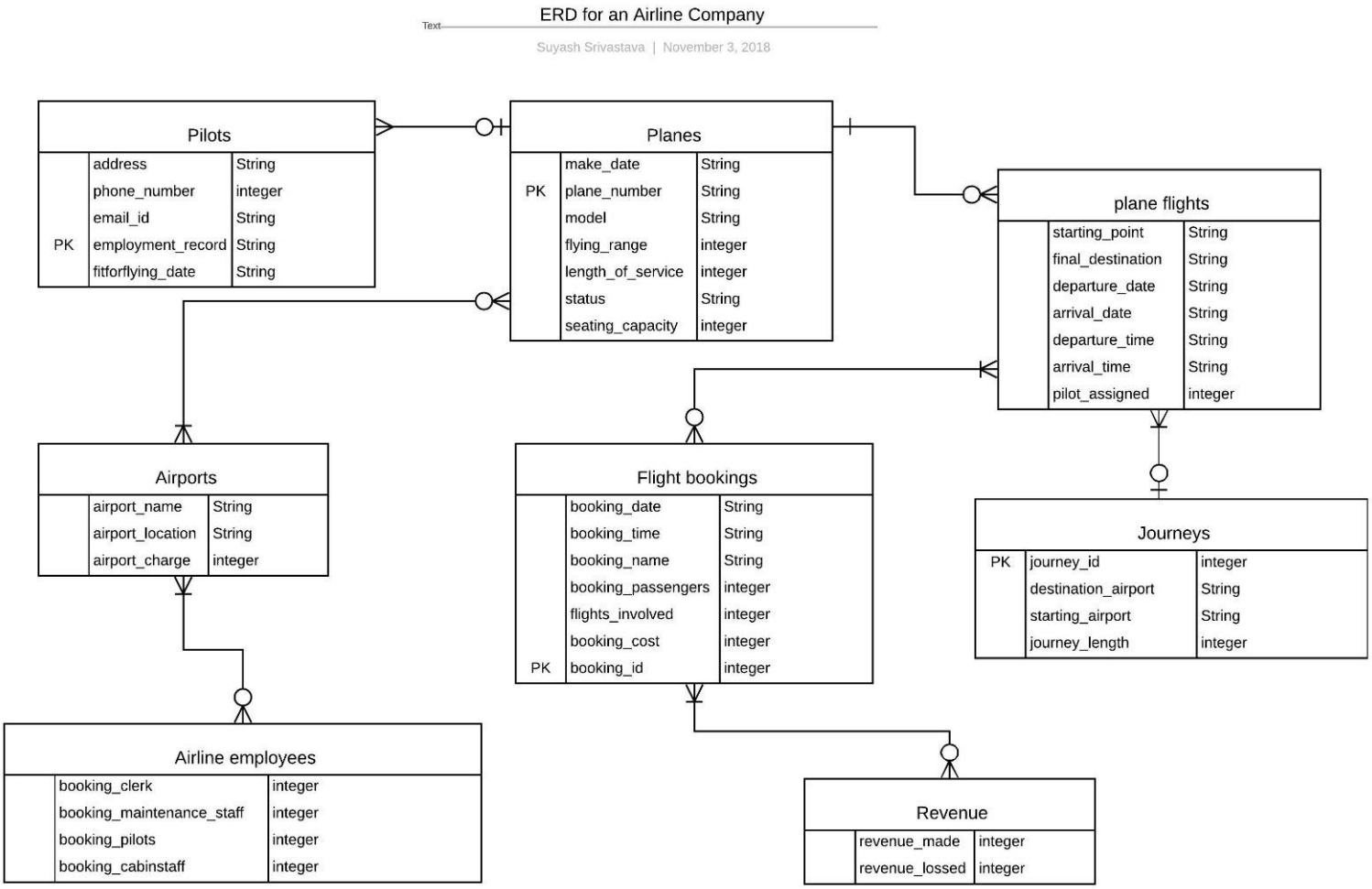
Contribution:

- Mukhtar Ismail: 100%
- Cruz Felix, Frank Erasmo: 100%
- Mikhdad Miah: 100%
- Suyash Srivastava : 0%

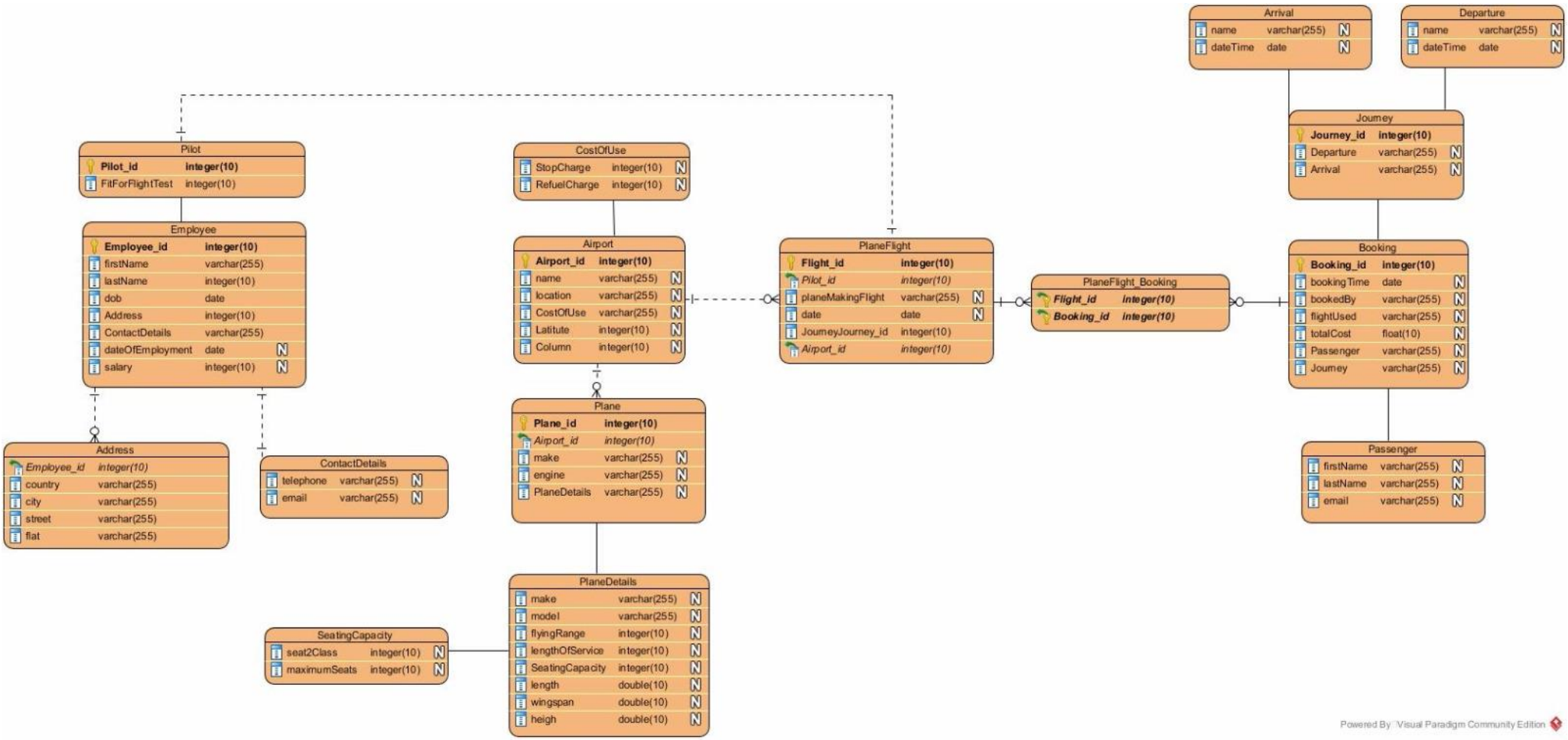
Assumptions

1. Each pilot “fit for fly test” is valid for 1 year
2. Revenue currency is in pounds
3. Each flight has cabin staff
4. Journey should be in the booking entity
5. Each plane has a base airport
6. A plane can stop at multiple airports, but belongs to one airport
7. For the revenue calculations, the salary and airport costs are on a yearly basis
8. Airport company is based in one single airport
9. Employees are based in a certain airport
10. There are two airports - one for departure and one for arrival
11. Maximum of 5 passengers per booking
12. Employees (pilots, cabin staff and maintenance staff) work for airline company and not for the airport

ER Diagram SQL



ER Diagram (NoSQL MongoDB design)



Database design (relational approach)

Relational database design consists of the following tables, representing subject area, according to the following logic:

1. **Pilot table:** details of the id, firstname, surname, dateOfBirth, contact details, date of fit for fly flying test, date of employment and salary

Name of field	Data type	Comment
_id	int (primary key)	
firstName	varchar	
Surname	varchar	
dateOfBirth	DateTime	
Address	varchar	This Object column contains columns: country (String), city (String), street (String) and flat (Int).
Contact details	object	This Object column contains columns: telephone (String because of different possible formats) and email (String)
dateOfFitForFlyingTest	DateTime	
dateOfEmployment	DateTime	
Salary	int	

Planes table: details of their make, model, flying range (with a full fuel load), length of service, status(i.e. Working, being repaired or upgraded etc), seating capacity

Name of field	Data type	Comment
planeID	int(primary key)	
planes	Object	This Object column contains columns: model(varchar), flying range(int), lengthofservice(int), status(varchar); seatingcapacity - seat2class(int), maximumseats(int); length(int), wingspan(int) and height(int).

In each airplane there are two different seating options, seat2Class and maximum seats.

planeFlights collection: details including flightID, planemakingflight, pilotID, startingpoint, finaldestination, dateTimes : departure and arrival.

Name of field	**Data type**	**Comment**
flightID	int(primary key)	
pilotID	int(foreign key)	
planemakingflight	varchar(foreign key)	
startingpoint	varchar	
finaldestination	Varchar	
dateTimes	Object	This object column contains: departure(DateTime) and arrival(DateTime)

flightBookings collection: details including bookingTime, bookedBy, flightsUsed, totalcostofbooking, Passengers: name, email and number.

Name of field	Data Type	Comment
bookingID	int(primary key)	
bookingTime	DateTime	
bookedBy	Varchar(foreign key)	
flightsUsed	Varchar(foreign key)	
totalcostbooking	Int	
Passengers	Object(foreign key)	This object column contains: name(varchar), email(varchar) and number(varchar).

Airports: including the name, location, cost of use (as a simplification, you can assume that each airport charges a fixed hourly rate for the length of time a plane stops at an airport, plus a further fixed charge for refueling, both of these charges will vary from airport to airport)

Name of field	Data Type	Comment
airportID	int(primary key)	
airportName	Varchar	

location	Varchar	
costOfUse	Object	This object column contains columns: planestops(int) and refuelling(int)

Journeys: including journey id, starting airport, destination airport, journey length (in kilometers)

Name of field	Data Type	Comment
journeyID(foreign key)	int(primary key)	
startingAirport	Varchar(foreign key)	
destinationAirport	Varchar(foreign key)	
journeyLength	int	

Airline employees: booking clerks - details in the collection include: id, first name, surname, date of birth, address - country, city, street, flat, contact details - telephone, email and salary

Name of field	Data Type	Comment
_id	int(primary key)	
firstName	varchar	
surname	varchar	
dateOfBirth	DateTime	
address	Object	This object column contains columns: country(varchar), city(varchar), street(varchar) and flat(varchar)
contactDetails	object	This object column contains columns: telephone(varchar) and email(varchar)
salary	Int	

Airline employees: maintenance staff - details in the collection include: id, first name, surname, date of birth, address - country, city, street, flat, contact details - telephone, email and salary

Name of field	Data Type	Comment
_id	int(primary key)	
firstName	varchar	

surname	varchar	
dateOfBirth	DateTime	
address	Object	This object column contains columns: country(varchar), city(varchar), street(varchar) and flat(varchar)
contactDetails	object	This object column contains columns: telephone(varchar) and email(varchar)
salary	Int	

Airline employees: details in the collection include: id, first name, surname, date of birth, address - country, city, street, flat, contact details - telephone, email and salary

Name of field	Data Type	Comment
_id	int(primary key)	
firstName	varchar	
surname	varchar	
dateOfBirth	DateTime	
address	Object	This object column contains columns: country(varchar), city(varchar), street(varchar) and flat(varchar)
contactDetails	object	This object column contains columns: telephone(varchar) and email(varchar)
salary	Int	

NoSQL database design

The NoSQL database was built using MongoDB RDBMS. A MongoDB script create_db.js inserts the initial documents into the collections. The parts of this script and explanation of its work and created collections are provided below.

Before creating the new database it should be ensured that there is no database with the same name:
conn = new Mongo(); db = conn.getDB('airlineCompany');

```
// drop the previous database with the same name if exists
 db.dropDatabase();
```

Pilot collection:

Name of field	Data type	Comment
_id	int (primary key)	
firstName	varchar	
Surname	varchar	
dateOfBirth	DateTime	
Address	varchar	This Object field contains fields: country (String), city (String), street (String) and flat (Int).
Contact details	object	This Object field contains fields: telephone (String because of different possible formats) and email (String)
Salary	int	
dateOfFitForFlyingTest	DateTime	
dateOfEmployment	DateTime	

Relationships:

Pilot - One-to-One relationship with embedded documents.

```
db.Pilot.insert({
    _id: 123456,
    firstName : "Mukhtar",
    lastName : "Ismail",
    dateOfBirth : new Date(1995,2,6,12,20),
    address : {
        country : "UK",
```

```
      city : "Birmingham",
      street : "Morley Street",
      flat : 15 },
   contactDetails : {
      telephone : "+44 02075455410",
      email : "mukhz32@gmail.com" },
   dateOfEmployment : new Date(2016,2,15,5),
   dateOfFitForFlyingTest: new Date(2016,2,10,0),
   salary: 100000
})
```

Plane collection:

Name of field	Data type	Comment
planes	Object	This Object field contains fields: model(varchar), flyingrange(int), lengthofservice(int), status(varchar), seatingcapacity - seat2class(int), maximumseats(int); length(int), wingspan(int) and height(int).
make	varchar	
engine	varchar	

Relationships

- One to many relationship with embedded document

```
db.Plane.insert({
   make: "Boeing",
         engine: "CFM-56",
   planes :[

            {
            model: "737-800",
            flyingrange: 5562,
            lengthofservice: 30,
            status: "working",
            seatingcapacity:{
               seat2CLass: 126,
               maximumSeats: 149},
            length: 33.6,
            wingspan: 38.5,
            height: 12.5,

         },

 {
```

```
            model: "737-700",
            flyingrange: 5500,
            lengthofservice: 27,
            status : "Beingrepaired",
            seatingcapacity:{
               seat2Class : 162,
               maximumSeats : 189},
            length: 39.5,
            wingspan: 38.5,
            height: 12.5,
         },
db.Plane.insert({
   make: "Boeing",
         engine: "CFM-56",
   planes :[

            {
            model: "737-800",
            flyingrange: 5562,
            lengthofservice: 30,
            status: "working",
            seatingcapacity:{
               seat2CLass: 126,
               maximumSeats: 149},
            length: 33.6,
            wingspan: 38.5,
            height: 12.5,

         },

 {
            model: "737-700",
            flyingrange: 5500,
            lengthofservice: 27,
            status : "Beingrepaired",
            seatingcapacity:{
               seat2Class : 162,
               maximumSeats : 189},
            length: 39.5,
            wingspan: 38.5,
            height: 12.5,
         },
```

```
db.Plane.insert({
   make: "Airbus",
   engine: "GP7000",
   planes:[
         {
            model: "A220-100",
            flyingrange: 5460,
            lengthofservice: 28,
            status: "Working",
            seatingcapacity: 116,
            length: 35.0,
            wingspan: 35.10,
            height: 11.50,
         }
```

// above is a sample of two makes of planes and their models.

Planeflights:

Name of field	Data type	Comment
pilotID	varchar	
startingpoint	varchar	
finaldestination	Varchar	
dateTimes	Object	This object field contains: departure(DateTime) and arrival(DateTime)
departure	DateTime	Not sure if row is needed
arrival	DateTime	Not sure if row is needed

```
db.PlaneFlight.insert({
    _id: "BA0107",
    startingpoint: "London Heathrow",
    finaldestination: "DXBDubai",
    pilotID: pilot1._id,
    dateTimes: {
        departure: new Date(2018,22,10,12,55),
        arrival: new Date (2018,22,10,23,5)}
})
```

Relationships

- One to One with embedded documents

flightBookings:

Name of field	Data Type	Comment
bookingTime	DateTime	
bookedBy	Varchar	
flightsUsed	Varchar	
totalcostbooking	Int	
Passengers	Object	This object field contains: name(varchar), email(varchar) and number(varchar).

```
db.FlightBooking.insert({
    bookingTime: new Date (2018,10,7,10,00),
    bookedBy: "Mithcel Jim",
    flightsUsed: flight4._id,
```

```
    totalcostofbooking: 3000,
    passengers: [
                        {
                                name: "Drw Mitchell",
                                email: "bobbykhaleque242@gmail.com",
                                number: "+44 751234524"

                        },

                        {
                                name: "Sten Fisr",
                                email: "stever505@gmail.com",
                                number: "+44 7562525255"
                        },

                        {
                                name: "Feix Mala",
                                email: "felilage453@gmail.com",
                                number: "+44 4712344474"
                        }
]
})
```

Relationship

- One to Many Relationship with embedded documents

Airports:

Name of field	Data Type	Comment
airportName	Varchar	
location	Varchar	
costOfUse	Object	This object field contains fields: planestops(int) and refuelling(int)

```
db.Airport.insert({
    airportName: "John F. Kennedy International Airport",
    location: "Queens,NewYork" ,
    costOfUse: {
        planestops: 8000,
        refuelling: 34000}
})

db.Airport.insert({
    airportName: "London Heathrow Airport",
    location: "London, England" ,
    costOfUse: {
        planestops: 10000,
        refuelling: 36000 }
})
```

Relationships

- One to One relationship with embedded documents

Journeys:

Name of field	Data Type	Comment
journeyID	int	
startingAirport	Varchar	
destinationAirport	Varchar	
journeyLength	int	

```
db.Journey.insert({
   journeyID: 9910,
   startingAirport:"Dubai International DXB",
   destinationAirport: "London Heathrow Airport",
   journeyLength: 5471.96
})

db.Journey.insert({
   journeyID: 8450,
   startingAirport:"London Heathrow Airport",
   destinationAirport:"Malaga Airport",
   journeyLength: 1676.12
})
```

Relationships:

One to One relationship with embedded documents

Employee:

Name of field	Data Type	Comment
_id	int	
firstName	varchar	
surname	varchar	
dateOfBirth	DateTime	
address	Object	This object field contains fields: country(varchar), city(varchar), street(varchar) and flat(varchar)
dateOfEmployment		
contactDetails	object	This object field contains fields: telephone(varchar) and email(varchar)
Employee type	varchar	

salary	Int	

```
db.Employee.insert(
{
	_id: 0342,
	firstName:"Carmela",
	surname: "Bella Mia",
	employeeType: "Cabin Staff",
	dateOfBirth : new Date(1992,8,2,3,15),
	dateOfEmployment : new Date(2005,2,1,8),
	address : {
		country : "UK",
		city : "London",
		street : "Italia Bella Avenue",
		flat : 27 },
	contactDetails : {
		telephone : "+44 7772453663",
		email : "carmelabm@hotmail.com"},
	salary: 48000
})
```

Relationships

- One to one with embedded documents

Airline employees: Pilots

Name of field	Data Type	Comment
pilotID	int	When inserting pilots, we use the id to link it to the pilots collection, so we can fetch the data from there.
employeeType	varchar	
dateOfEmployment	varchar	
salary	varchar	Will be used in the revenue calculation

```
var pilot1 = db.Pilot.findOne({_id:123456});
var pilot2 = db.Pilot.findOne({_id:234567});
var pilot3 = db.Pilot.findOne({_id:345678});

db.Employee.insert(
{
   _id: pilot1._id,
   employeeType: "Pilot",
   dateOfEmployment : pilot1.dateOfEmployment,
   salary: pilot1.salary
})
```

Relationships

- One to One with embedded documents

Queries

A set of queries are included below which demonstrate that the database has an appropriate design. These will include queries that will extract information which will typically be required for an airline company.

1) **The booking clerks will require all of the bookings for that day so they can allow passengers through.**

Query

```
db.FlightBooking.find({"bookingTime":{$gte:new Date(new Date().getFullYear(),new Date().getMonth(),new Date().getDate())}})
```

Result


```
{
   "_id" : ObjectId("5be49a1f0f9990b0aa2a786f"),
   "bookingTime" : ISODate("2018-11-08T10:00:00Z"),
   "bookedBy" : "Drew Mitchell",
   "flightsUsed" : "BA0107",
   "totalcostofbooking" : 3000,
   "passengers" : [
         {
               "name" : "Drew Mitchell",
               "email" : "bobbykhaleque242@gmail.com",
               "number" : "+44 751524524"
         },
         {
               "name" : "Steven Fisher",
               "email" : "stevenfisher505@gmail.com",
               "number" : "+44 7562525255"
         },
         {
               "name" : "Felix Malaga",
               "email" : "felixmalage453@gmail.com",
               "number" : "+44 4714844474"
         }
   ]
}
```

2) **The Fit For Flying Test should be taken once a year. This query will find all the pilots who will need to take their Fit For Flying Tests in the next month.**

Query

```
db.Pilot.find({"dateOfFitForFlyingTest":{$lte:new Date(new Date().setDate(new Date().getDate() - 334))}})
```

Result


```
{
    "_id" : 123456,
    "firstName" : "Mukhtar",
    "lastName" : "Ismail",
    "dateOfBirth" : ISODate("1995-03-06T12:20:00Z"),
    "address" : {
            "country" : "UK",
            "city" : "Birmingham",
            "street" : "Morley Street",
            "flat" : 15
    },
    "contactDetails" : {
            "telephone" : "+44 02075455410",
            "email" : "mukhz32@gmail.com"
    },
    "dateOfEmployment" : ISODate("2016-03-15T05:00:00Z"),
    "dateOfFitForFlyingTest" : ISODate("2016-03-10T00:00:00Z"),
    "salary" : 100000
}
{
    "_id" : 234567,
    "firstName" : "Juan",
    "lastName" : "Dolores",
    "dateOfBirth" : ISODate("1994-03-03T11:25:00Z"),
    "address" : {
            "country" : "UK",
            "city" : "London",
            "street" : "Oxford Street",
            "flat" : 3
    },
    "contactDetails" : {
            "telephone" : "+44 02084545121",
            "email" : "juanpilot101@gmail.com"
    },
    "dateOfEmployment" : ISODate("2015-04-12T01:00:00Z"),
    "dateOfFitForFlyingTest" : ISODate("2015-04-12T01:00:00Z"),
    "salary" : 125000
}
{
    "_id" : 345678,
    "firstName" : "Sadeq",
    "lastName" : "Rahman",
    "dateOfBirth" : ISODate("1994-02-03T10:00:00Z"),
    "address" : {
```

```
            "country" : "UK",
            "city" : "Wales",
            "street" : "Old Village",
            "flat" : 3
    },
    "contactDetails" : {
            "telephone" : "+44 02084545121",
            "email" : "mikesmitholdvillage@gmail.com"
    },
    "dateOfEmployment" : ISODate("2015-03-12T02:00:00Z"),
    "dateOfFitForFlyingTest" : ISODate("2015-03-11T01:00:00Z"),
    "salary" : 95000
}
```


3) **The most expensive bookings over a certain period of time could be used by marketing and advertising for sales strategies. This query will show the 5 most expensive bookings in the last month**

Query

```
db.flightBooking.aggregate([{$sort:{totalcostofbooking:-1}},{$limit:5}])
```

Result


```
{
    "_id" : ObjectId("5be49cc12ad13720b7fb41b5"),
    "bookingTime" : ISODate("2018-11-07T12:01:00Z"),
    "bookedBy" : "Jamie Smith",
    "flightsUsed" : "BA0117",
    "totalcostofbooking" : 11000,
    "passengers" : [
            {
                    "name" : "Khaz Barr",
                    "email" : "KB565@gmail.com",
                    "number" : "07515244851"
            },
            {
                    "name" : "Rajab Smith",
                    "email" : "RS67@gmail.com",
                    "number" : "07542405120"
            },
            {
                    "name" : "Jamie Smith",
                    "email" : "JS565@gmail.com",
                    "number" : "07515151331"
            }
    ]
}
{
    "_id" : ObjectId("5be49cb02ad13720b7fb41b3"),
    "bookingTime" : ISODate("2018-03-12T01:00:00Z"),
    "bookedBy" : "Johny Devine",
```

```
    "flightsUsed" : "BA0367",
    "totalcostofbooking" : 9900,
    "passengers" : [
            {
                    "name" : "Eva Rahman",
                    "email" : "ER65@gmail.com",
                    "number" : "07515424851"
            },
            {
                    "name" : "Shorob Rahman",
                    "email" : "SR67@gmail.com",
                    "number" : "07542234120"
            },
            {
                    "name" : "Johny Devine",
                    "email" : "JD5@gmail.com",
                    "number" : "07456151331"
            }
    ]
}
{
    "_id" : ObjectId("5be49c9e2ad13720b7fb41b1"),
    "bookingTime" : ISODate("2018-09-12T11:01:00Z"),
    "bookedBy" : "Jimmy Goliath",
    "flightsUsed" : "BA0117",
    "totalcostofbooking" : 9200,
    "passengers" : [
            {
                    "name" : "James Goliath",
                    "email" : "JG5@gmail.com",
                    "number" : "07234244851"
            },
            {
                    "name" : "Phoebe Goliath",
                    "email" : "PG67@gmail.com",
                    "number" : "07567805120"
            },
            {
                    "name" : "Sadeq Goliath",
                    "email" : "sadeqsmith565@gmail.com",
                    "number" : "07515156541"
            }
    ]
}
{
    "_id" : ObjectId("5be49c942ad13720b7fb41b0"),
    "bookingTime" : ISODate("2018-09-07T11:01:00Z"),
    "bookedBy" : "Sumaya Deen",
    "flightsUsed" : "BA0127",
    "totalcostofbooking" : 8000,
    "passengers" : [
            {
                    "name" : "James Deen",
                    "email" : "JD@gmail.com",
                    "number" : "07515234851"
```

```
        },
        {
                "name" : "Mikhdad Deen",
                "email" : "MD7@gmail.com",
                "number" : "07542405120"
        },
        {
                "name" : "Karen Smith",
                "email" : "KS@gmail.com",
                "number" : "07578951331"
        }
    ]
}
{
    "_id" : ObjectId("5be49cb92ad13720b7fb41b4"),
    "bookingTime" : ISODate("2018-05-11T00:15:00Z"),
    "bookedBy" : "Mikeala Dolo",
    "flightsUsed" : "BA0127",
    "totalcostofbooking" : 7800,
    "passengers" : [
        {
                "name" : "Mikeala Dolo",
                "email" : "Mikaela565@gmail.com",
                "number" : "07515247651"
        },
        {
                "name" : "Mikhdad Smith",
                "email" : "MikhdadSmith67@gmail.com",
                "number" : "07542404350"
        },
        {
                "name" : "Sadeg Smith",
                "email" : "sadeqsmith565@gmail.com",
                "number" : "07515153211"
        }
    ]
}
```


4) **Employees who have been with the company for the longest period of time may be due a loyalty bonus. This query will show the 3 employees who have been with the company the longest.**

Query

```
db.Employee.find().sort({dateOfEmployment:1}).limit(3);
```

Result


```
{
    "_id" : 226,
    "firstName" : "Carmela",
    "surname" : "Bella Mia",
    "employeeType" : "Cabin Staff",
```

```
    "dateOfBirth" : ISODate("1992-09-02T02:15:00Z"),
    "dateOfEmployment" : ISODate("2005-03-01T08:00:00Z"),
    "address" : {
            "country" : "UK",
            "city" : "London",
            "street" : "Italia Bella Avenue",
            "flat" : 27
    },
    "contactDetails" : {
            "telephone" : "+44 7772453663",
            "email" : "carmelabm@hotmail.com"
    },
    "salary" : 48000
}
{
    "_id" : 225,
    "firstName" : "Rodrigo",
    "surname" : "Sinfuentes",
    "employeeType" : "Cabin Staff",
    "dateOfBirth" : ISODate("1987-07-10T10:45:00Z"),
    "dateOfEmployment" : ISODate("2010-03-01T06:00:00Z"),
    "address" : {
            "country" : "UK",
            "city" : "Manchester City",
            "street" : "Front Road",
            "flat" : 14
    },
    "contactDetails" : {
            "telephone" : "+44 7772453663",
            "email" : "rodrigosf08@hotmail.com"
    },
    "salary" : 48000
}
{
    "_id" : 8458,
    "firstName" : "Mikhdad",
    "surname" : "David",
    "employeeType" : "Maintenence Staff",
    "dateOfBirth" : ISODate("1989-01-08T21:45:00Z"),
    "dateOfEmployment" : ISODate("2014-03-13T05:00:00Z"),
    "address" : {
            "country" : "UK",
            "city" : "London",
            "street" : "Seven Sister Avenue",
            "flat" : 97
    },
    "contactDetails" : {
            "telephone" : "+44 7714958135",
            "email" : "mdavid@gmail.com"
    },
    "salary" : 30000
}
```

5) **When pilots have passed there Fit For Flying Test, this field will need to be updated.**

Query

```
var tempDateForFitForFlyingTest = new Date();
db.Pilot.update({_id:
pilot1._id},{$set:{dateOfFitForFlyingTest:tempDateForFitForFlyingTest}})
```

Result

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

{
    "_id" : 123456,
    "firstName" : "Mukhtar",
    "lastName" : "Ismail",
    "dateOfBirth" : ISODate("1995-03-06T12:20:00Z"),
    "address" : {
            "country" : "UK",
            "city" : "Birmingham",
            "street" : "Morley Street",
            "flat" : 15
    },
    "contactDetails" : {
            "telephone" : "+44 02075455410",
            "email" : "mukhz32@gmail.com"
    },
    "dateOfEmployment" : ISODate("2016-03-15T05:00:00Z"),
    "dateOfFitForFlyingTest" : ISODate("2018-11-08T20:41:05.124Z"),
    "salary" : 100000
}
```

6) The total amount of money made from bookings will be required to calculate the revenue. This information will also be useful for marketing purposes. This query will return the total cost of the bookings in the last year, and it will store it in a variable so it can be used in the revenue calculation later on.

Query

```
var totalBooking =  db.FlightBooking.aggregate( [
{ $match:{'bookingTime':{$gte:   ISODate("2018-00-01T00:00:00.000Z")},'bookingTime':{$lte:
ISODate("2018-11-31T23:59:59.000Z")}}},
{ $group : { _id : null, totalBooking: { $sum: "$totalcostofbooking" } } },] ).toArray()
```

Result

```
[ { "_id" : null, "totalBooking" : 64558 } ]
```

7) **The total airport costs will also be required for the revenue calculation. This query will return the sum of the plane stops and refuelling costs for all of the airports. The result will be stored in a variable so it can be used in the revenue calculation later on.**

Query

```
var totalCost = db.Airport.aggregate( [
{ $group : { _id : null, totalCosts: { $sum: { $add : [ '$costOfUse.planestops',
'$costOfUse.refuelling' ] } } } },] ).toArray()
```

Result

```
[ { "_id" : null, "totalCosts" : 166600 } ]
```

8) **The total salary for all the employees will also be required for the revenue calculation. This query will return the sum of all the employees who work for the company. The result will be stored in a variable so it can be used in the revenue calculation later on.**

Query

```
var totalSalary = db.Employee.aggregate( [
{ $group : { _id : null, totalSalary: { $sum: "$salary" } } },] ).toArray()
```

Result

```
[ { "_id" : null, "totalSalary" : 566000 } ]
```

9) The total revenue for the year will be displayed by the query below.

Query - Inserting required fields

```
db.Revenue.insert({
        _id:12987,
        totalBooking: totalBooking[0].totalBooking,
        totalCost: totalCost[0].totalCosts,
        totalSalary: totalSalary[0].totalSalary
})
```

Result

```
WriteResult({ "nInserted" : 1 })

{ "_id" : 12987, "totalBooking" : 64558, "totalCost" : 166600, "totalSalary" : 566000 }
```

Query - Calculating Revenue

```
db.Revenue.aggregate( [
{ $project : {_id: 0 ,totalRevenue: {$subtract: [{$subtract:
["$totalBooking","$totalCost"]},"$totalSalary"]}}}])
```

Result

```
{ "totalRevenue" : -668042 }
```

Usage of Performance monitoring tools

Explain() :

The explain command returns information on the query plan for the following operations: aggregate(), count(), distinct(), find(), remove(), and update() methods.

The verbosity of the *db.collection.explain()* and the amount of information returned depend on the verbosity mode. These mode affects the behaviour of explain(). They are:

- "queryPlanner"
- "executionStats"
- "allPlansExecution"

The FlightBooking collection is assumed to be one of the most frequently used in the current project. A new document is added to the collection after each booking made in the Airplane Company.

Evaluate the performance of a query

The following query used retrieves documents belonging to the day 07th of November 2017.

```
db.FlightBooking.find({ bookingTime:{
   $gte: ISODate("2018-11-07T00:00:00.000Z"),
   $lte: ISODate("2018-11-07T23:59:59.000Z")

}})
```

The query returns four documents (all docmuents inserted in this collection for this example). The method explain() with parameter ***executionStats*** runs the query optimizer to choose the winning query plan. The method executes the winning plan and returns statistics describing executing of the winning plan. This mode also details the rejected plans.

Adding explain("executionStats")method:

```
db.FlightBooking.find({ bookingTime:{
   $gte: ISODate("2018-11-07T00:00:00.000Z"),
   $lte: ISODate("2018-11-07T23:59:59.000Z")

}}).explain("executionStats")
```

Terminal output:
This query with no idex number returns following:


```
{
	"queryPlanner" : {
		"plannerVersion" : 1,
		"namespace" : "AirplaneCompany.FlightBooking",
		"indexFilterSet" : false,
		"parsedQuery" : {
			"$and" : [
				{
					"bookingTime" : {
						"$lte" : ISODate("2018-11-07T23:59:59Z")
					}
				},
				{
					"bookingTime" : {
						"$gte" : ISODate("2018-11-07T00:00:00Z")
					}
				}
			]
		},
		"winningPlan" : {
			"stage" : "COLLSCAN",
			"filter" : {
				"$and" : [
					{
						"bookingTime" : {
							"$lte" : ISODate("2018-11-07T23:59:59Z")
						}
					},
					{
						"bookingTime" : {
							"$gte" : ISODate("2018-11-07T00:00:00Z")
						}
					}
				]
			},
			"direction" : "forward"
		},
		"rejectedPlans" : [ ]
	},
	"executionStats" : {
		"executionSuccess" : true,
		"nReturned" : 3,
		"executionTimeMillis" : 0,
		"totalKeysExamined" : 0,
		"totalDocsExamined" : 10,
		"executionStages" : {
			"stage" : "COLLSCAN",
			"filter" : {
				"$and" : [
					{
						"bookingTime" : {
							"$lte" : ISODate("2018-11-07T23:59:59Z")
```

```
                                }
                        },
                        {
                                "bookingTime" : {
                                        "$gte" : ISODate("2018-11-07T00:00:00Z")
                                }
                        }
                ]
        },
        "nReturned" : 3,
        "executionTimeMillisEstimate" : 0,
        "works" : 12,
        "advanced" : 3,
        "needTime" : 8,
        "needYield" : 0,
        "saveState" : 0,
        "restoreState" : 0,
        "isEOF" : 1,
        "invalidates" : 0,
        "direction" : "forward",
        "docsExamined" : 10
    }
  },
  "serverInfo" : {
        "host" : "Frank-Cf",
        "port" : 27017,
        "version" : "4.0.2",
        "gitVersion" : "fc1573ba18aee42f97a3bb13b67af7d837826b47"
  },
  "ok" : 1
}
```


From the output, it can be said that:

- **queryPlanner.winningPlan.stage** shows **COLLSCAN**, meaning that the query planner selected a collection scan (no index was used)
- **executionStats.nReturned** displays 3, indicating the number of documents that match the query condition.
- **executionStats.totalDocsExamined** displays 10, indicating the number of documents scanned (all documents in FlightBooking collection for this example) to find the four matching documents.

The difference between the number of matching and scanned documents might suggest that the query would benefit from the use of an index (1). Therefore, an index on the **bookingTime** field is added:

db.FlightBooking.createIndex({ bookingTime: 1})

Having as a summary (terminal output)


```
{
        "createdCollectionAutomatically" : false,
```

```
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

Using the explain() method with the query:

```
> db.FlightBooking.find({ bookingTime:{
...     $gte: ISODate("2018-11-07T00:00:00.000Z"),
...     $lte: ISODate("2018-11-07T23:59:59.000Z")
...
... }}).explain("executionStats")
```

The explain() method returns the following result now:

```
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "AirplaneCompany.FlightBooking",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "$and" : [
                                {
                                        "bookingTime" : {
                                                "$lte" : ISODate("2018-11-07T23:59:59Z")
                                        }
                                },
                                {
                                        "bookingTime" : {
                                                "$gte" : ISODate("2018-11-07T00:00:00Z")
                                        }
                                }
                        ]
                },
                "winningPlan" : {
                        "stage" : "FETCH",
                        "inputStage" : {
                                "stage" : "IXSCAN",
                                "keyPattern" : {
                                        "bookingTime" : 1
                                },
                                "indexName" : "bookingTime_1",
                                "isMultiKey" : false,
                                "multiKeyPaths" : {
                                        "bookingTime" : [ ]
                                },
                                "isUnique" : false,
                                "isSparse" : false,
                                "isPartial" : false,
                                "indexVersion" : 2,
                                "direction" : "forward",
                                "indexBounds" : {
                                        "bookingTime" : [
                                                "[new Date(1541548800000), new Date(1541635199000)]"
```

```
                        ]
                    }
                }
            },
            "rejectedPlans" : [ ]
    },
    "executionStats" : {
            "executionSuccess" : true,
            "nReturned" : 3,
            "executionTimeMillis" : 377,
            "totalKeysExamined" : 3,
            "totalDocsExamined" : 3,
            "executionStages" : {
                    "stage" : "FETCH",
                    "nReturned" : 3,
                    "executionTimeMillisEstimate" : 0,
                    "works" : 4,
                    "advanced" : 3,
                    "needTime" : 0,
                    "needYield" : 0,
                    "saveState" : 0,
                    "restoreState" : 0,
                    "isEOF" : 1,
                    "invalidates" : 0,
                    "docsExamined" : 3,
                    "alreadyHasObj" : 0,
                    "inputStage" : {
                            "stage" : "IXSCAN",
                            "nReturned" : 3,
                            "executionTimeMillisEstimate" : 0,
                            "works" : 4,
                            "advanced" : 3,
                            "needTime" : 0,
                            "needYield" : 0,
                            "saveState" : 0,
                            "restoreState" : 0,
                            "isEOF" : 1,
                            "invalidates" : 0,
                            "keyPattern" : {
                                    "bookingTime" : 1
                            },
                            "indexName" : "bookingTime_1",
                            "isMultiKey" : false,
                            "multiKeyPaths" : {
                                    "bookingTime" : [ ]
                            },
                            "isUnique" : false,
                            "isSparse" : false,
                            "isPartial" : false,
                            "indexVersion" : 2,
                            "direction" : "forward",
                            "indexBounds" : {
                                    "bookingTime" : [
                                            "[new Date(1541548800000), new Date(1541635199000)]"
                                    ]
```

```
                        },
                        "keysExamined" : 3,
                        "seeks" : 1,
                        "dupsTested" : 0,
                        "dupsDropped" : 0,
                        "seenInvalidated" : 0
                }
        }
    },
    "serverInfo" : {
        "host" : "Frank-Cf",
        "port" : 27017,
        "version" : "4.0.2",
        "gitVersion" : "fc1573ba18aee42f97a3bb13b67af7d837826b47"
    },
    "ok" : 1
}
```


From the output, it can be said that:

- **queryPlanner.winningPlan.inputStage.stage** displays **IXSCAN**, indicating that index keys were scanned.
- **executionStats.nReturned** displays 3, indicating the number of documents that match the query condition.
- **executionStats.totalKeysExamined** displays 3, indicating the number of index entries scanned.

- **executionStats.totalDocsExamined** displays 3, indicating the number of documents scanned.

Conclusion:

With an index, the query scanned 3 index entries and 3 documents to return 3 matching documents. On the other hand, without the index, the query has to scan the whole collection (10 documents inserted in this example). Therefore, it has been proved that efficiency is improved by creating an index for this query.

To have a better view about creating an index it is a good idea. It will be randomly created and added to the FlightBoooking collections two thousand bookings. Then, it will be run the find query with and without index.

The code below adds 1000 random bookings.

```
Connect = new Mongo();

Db = cononect.getDB('AirlineCompany); db
```

//db.dropDatabase();//**If is there are any databases created with the same name**

```
function randomInt(min,max){
   return Math.floor(Math.random()*(max-min+1)) +min;
}
```

```
var pilots = db.Pilot.find();
var flights = db.PlaneFlight.find();
var booked = db.FlightBooking.find();

var randBookings = [];

for(var i=0;i<1000;i++){
var date=newDate(2018,randomInt(5,11),randomInt(1,31),randomInt(0,23),randomInt(0,59));
var ranPilot = pilots[randomInt(0,pilots.length()-1)];
   var ranFlight = flights[randomInt(0,flights.length()-1)];
   var randBooked = booked[randomInt(0,booked.length()-1)];
   var randCost = randomInt(1000,3500);

   randBookings.push({
      'bookingTime': date,
      'bookedBy': randBooked.bookedBy,
      'flightsUsed': ranFlight._id,
      'totalcostofbooking': randCost,
      'passengers': randBooked.passengers
   });
}
db.FlightBooking.insertMany(randBookings);
```

Here is the following query with no index:

```
> db.FlightBooking.find({ bookingTime:{
...     $gte: ISODate("2018-11-07T00:00:00.000Z"),
...     $lte: ISODate("2018-11-07T23:59:59.000Z")
...
... }}).explain("executionStats")
```

After adding and running *explain("executionStats")* method:


```
{
      "queryPlanner" : {
            "plannerVersion" : 1,
            "namespace" : "AirplaneCompany.FlightBooking",
            "indexFilterSet" : false,
            "parsedQuery" : {
                  "$and" : [
                        {
                              "bookingTime" : {
                                    "$lte" : ISODate("2018-11-07T23:59:59Z")
                              }
                        },
                        {
                              "bookingTime" : {
                                    "$gte" : ISODate("2018-11-07T00:00:00Z")
```

```
                        }
                    }
                ]
        },
        "winningPlan" : {
            "stage" : "COLLSCAN",
            "filter" : {
                "$and" : [
                    {
                        "bookingTime" : {
                            "$lte" : ISODate("2018-11-07T23:59:59Z")
                        }
                    },
                    {
                        "bookingTime" : {
                            "$gte" : ISODate("2018-11-07T00:00:00Z")
                        }
                    }
                ]
            },
            "direction" : "forward"
        },
        "rejectedPlans" : [ ]
    },
    "executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 8,
        "executionTimeMillis" : 0,
        "totalKeysExamined" : 0,
        "totalDocsExamined" : 1038,
        "executionStages" : {
            "stage" : "COLLSCAN",
            "filter" : {
                "$and" : [
                    {
                        "bookingTime" : {
                            "$lte" : ISODate("2018-11-07T23:59:59Z")
                        }
                    },
                    {
                        "bookingTime" : {
                            "$gte" : ISODate("2018-11-07T00:00:00Z")
                        }
                    }
                ]
            },
            "nReturned" : 8,
            "executionTimeMillisEstimate" : 0,
            "works" : 1040,
            "advanced" : 8,
            "needTime" : 1031,
            "needYield" : 0,
            "saveState" : 8,
            "restoreState" : 8,
            "isEOF" : 1,
```

```
                "invalidates" : 0,
                "direction" : "forward",
                "docsExamined" : 1038
            }
        },
        "serverInfo" : {
            "host" : "Frank-Cf",
            "port" : 27017,
            "version" : "4.0.2",
            "gitVersion" : "fc1573ba18aee42f97a3bb13b67af7d837826b47"
        },
        "ok" : 1
}
```


The results above can be said that:

- **queryPlanner.winningPlan.stage** shows **COLLSCAN**, meaning that the query planner selected a collection scan (no index was used)
- **executionStats.nReturned** displays 8, indicating the number of documents that match the query condition.
- **executionStats.totalDocsExamined** displays 1038, indicating the number of documents scanned (all documents in FlightBooking collection for this example) to find the eight matching documents.

Query with index:

db.FlightBooking.createIndex({ bookingTime: 1})

```
> db.FlightBooking.find({ bookingTime:{
...     $gte: ISODate("2018-11-07T00:00:00.000Z"),
...     $lte: ISODate("2018-11-07T23:59:59.000Z")
...
... }}).explain("executionStats")
```


```
{
        "queryPlanner" : {
            "plannerVersion" : 1,
            "namespace" : "AirplaneCompany.FlightBooking",
            "indexFilterSet" : false,
            "parsedQuery" : {
                "$and" : [
                    {
                        "bookingTime" : {
                            "$lte" : ISODate("2018-11-07T23:59:59Z")
                        }
                    },
                    {
                        "bookingTime" : {
                            "$gte" : ISODate("2018-11-07T00:00:00Z")
                        }
                    }
                ]
```

```
        },
        "winningPlan" : {
                "stage" : "FETCH",
                "inputStage" : {
                        "stage" : "IXSCAN",
                        "keyPattern" : {
                                "bookingTime" : 1
                        },
                        "indexName" : "bookingTime_1",
                        "isMultiKey" : false,
                        "multiKeyPaths" : {
                                "bookingTime" : [ ]
                        },
                        "isUnique" : false,
                        "isSparse" : false,
                        "isPartial" : false,
                        "indexVersion" : 2,
                        "direction" : "forward",
                        "indexBounds" : {
                                "bookingTime" : [
                                        "[new Date(1541548800000), new Date(1541635199000)]"
                                ]
                        }
                }
        },
        "rejectedPlans" : [ ]
},
"executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 8,
        "executionTimeMillis" : 147,
        "totalKeysExamined" : 8,
        "totalDocsExamined" : 8,
        "executionStages" : {
                "stage" : "FETCH",
                "nReturned" : 8,
                "executionTimeMillisEstimate" : 0,
                "works" : 9,
                "advanced" : 8,
                "needTime" : 0,
                "needYield" : 0,
                "saveState" : 0,
                "restoreState" : 0,
                "isEOF" : 1,
                "invalidates" : 0,
                "docsExamined" : 8,
                "alreadyHasObj" : 0,
                "inputStage" : {
                        "stage" : "IXSCAN",
                        "nReturned" : 8,
                        "executionTimeMillisEstimate" : 0,
                        "works" : 9,
                        "advanced" : 8,
                        "needTime" : 0,
                        "needYield" : 0,
```

```
                        "saveState" : 0,
                        "restoreState" : 0,
                        "isEOF" : 1,
                        "invalidates" : 0,
                        "keyPattern" : {
                                "bookingTime" : 1
                        },
                        "indexName" : "bookingTime_1",
                        "isMultiKey" : false,
                        "multiKeyPaths" : {
                                "bookingTime" : [ ]
                        },
                        "isUnique" : false,
                        "isSparse" : false,
                        "isPartial" : false,
                        "indexVersion" : 2,
                        "direction" : "forward",
                        "indexBounds" : {
                                "bookingTime" : [
                                        "[new Date(1541548800000), new Date(1541635199000)]"
                                ]
                        },
                        "keysExamined" : 8,
                        "seeks" : 1,
                        "dupsTested" : 0,
                        "dupsDropped" : 0,
                        "seenInvalidated" : 0
                }
        }
    },
    "serverInfo" : {
            "host" : "Frank-Cf",
            "port" : 27017,
            "version" : "4.0.2",
            "gitVersion" : "fc1573ba18aee42f97a3bb13b67af7d837826b47"
    },
    "ok" : 1
}
```


From the output, it can be said that:

- **queryPlanner.winningPlan.inputStage.stage** displays **IXSCAN**, indicating that index keys were scanned.
- **executionStats.nReturned** displays 8, indicating the number of documents that match the query condition.
- **executionStats.totalKeysExamined** displays 8, indicating the number of index entries scanned.

- **executionStats.totalDocsExamined** displays 8, indicating the number of documents scanned.

Conclusion:

From the results above, it can be said that for a larger collection the difference between matched and scanned documents using a query with and without index is much more notable. The query with no index returns, 8 matching documents after scanning the whole collection (1038 documents after creating them randomly). On contrary, creating an index improves the performance when scanning documents. The query with index scanned 8 index entries and 8 documents to return 8 matching documents.

To conclude, Indexes help improve sorting performance and query, On the other hand, the cost is the amount of time processing operations. Therefore, when creating indexes and evaluating them, ensure that the query is actually using these indexes

Evaluating if the query uses indexes

The explain() and update() methods are used in the query with and without indexes:

```
> db.FlightBooking.explain("executionStats").update(
...     {"bookingTime": ISODate("2018-09-21T08:22:00.000Z")},
...     {$set:{"bookingTime": ISODate("2018-09-21T02:15:00.000Z")}}
... )
```

Query with no index:


```
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "AirplaneCompany.FlightBooking",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "bookingTime" : {
                                "$eq" : ISODate("2018-09-21T08:22:00Z")
                        }
                },
                "winningPlan" : {
                        "stage" : "UPDATE",
                        "inputStage" : {
                                "stage" : "COLLSCAN",
                                "filter" : {
                                        "bookingTime" : {
                                                "$eq" : ISODate("2018-09-21T08:22:00Z")
                                        }
                                },
                                "direction" : "forward"
                        }
                },
                "rejectedPlans" : [ ]
        },
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 0,
                "executionTimeMillis" : 29,
                "totalKeysExamined" : 0,
                "totalDocsExamined" : 1010,
                "executionStages" : {
                        "stage" : "UPDATE",
```

```
                "nReturned" : 0,
                "executionTimeMillisEstimate" : 29,
                "works" : 1013,
                "advanced" : 0,
                "needTime" : 1012,
                "needYield" : 0,
                "saveState" : 8,
                "restoreState" : 8,
                "isEOF" : 1,
                "invalidates" : 0,
                "nMatched" : 0,
                "nWouldModify" : 0,
                "nInvalidateSkips" : 0,
                "wouldInsert" : false,
                "fastmodinsert" : false,
                "inputStage" : {
                        "stage" : "COLLSCAN",
                        "filter" : {
                                "bookingTime" : {
                                        "$eq" : ISODate("2018-09-21T08:22:00Z")
                                }
                        },
                        "nReturned" : 0,
                        "executionTimeMillisEstimate" : 29,
                        "works" : 1012,
                        "advanced" : 0,
                        "needTime" : 1011,
                        "needYield" : 0,
                        "saveState" : 8,
                        "restoreState" : 8,
                        "isEOF" : 1,
                        "invalidates" : 0,
                        "direction" : "forward",
                        "docsExamined" : 1010
                }
        }
        },
        "serverInfo" : {
                "host" : "Frank-Cf",
                "port" : 27017,
                "version" : "4.0.2",
                "gitVersion" : "fc1573ba18aee42f97a3bb13b67af7d837826b47"
        },
        "ok" : 1
}
```


Query with index created:


```
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "AirplaneCompany.FlightBooking",
                "indexFilterSet" : false,
                "parsedQuery" : {
```

```
                "bookingTime" : {
                        "$eq" : ISODate("2018-09-21T08:22:00Z")
                }
        },
        "winningPlan" : {
                "stage" : "UPDATE",
                "inputStage" : {
                        "stage" : "FETCH",
                        "inputStage" : {
                                "stage" : "IXSCAN",
                                "keyPattern" : {
                                        "bookingTime" : 1
                                },
                                "indexName" : "bookingTime_1",
                                "isMultiKey" : false,
                                "multiKeyPaths" : {
                                        "bookingTime" : [ ]
                                },
                                "isUnique" : false,
                                "isSparse" : false,
                                "isPartial" : false,
                                "indexVersion" : 2,
                                "direction" : "forward",
                                "indexBounds" : {
                                        "bookingTime" : [
                                                "[new Date(1537518120000), new Date(1537518120000)]"
                                        ]
                                }
                        }
                }
        },
        "rejectedPlans" : [ ]
    },
    "executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 0,
        "executionTimeMillis" : 27,
        "totalKeysExamined" : 1,
        "totalDocsExamined" : 1,
        "executionStages" : {
                "stage" : "UPDATE",
                "nReturned" : 0,
                "executionTimeMillisEstimate" : 0,
                "works" : 2,
                "advanced" : 0,
                "needTime" : 1,
                "needYield" : 0,
                "saveState" : 0,
                "restoreState" : 0,
                "isEOF" : 1,
                "invalidates" : 0,
                "nMatched" : 1,
                "nWouldModify" : 1,
                "nInvalidateSkips" : 0,
```

```
                "wouldInsert" : false,
                "fastmodinsert" : false,
                "inputStage" : {
                        "stage" : "FETCH",
                        "nReturned" : 1,
                        "executionTimeMillisEstimate" : 0,
                        "works" : 1,
                        "advanced" : 1,
                        "needTime" : 0,
                        "needYield" : 0,
                        "saveState" : 1,
                        "restoreState" : 1,
                        "isEOF" : 0,
                        "invalidates" : 0,
                        "docsExamined" : 1,
                        "alreadyHasObj" : 0,
                        "inputStage" : {
                                "stage" : "IXSCAN",
                                "nReturned" : 1,
                                "executionTimeMillisEstimate" : 0,
                                "works" : 1,
                                "advanced" : 1,
                                "needTime" : 0,
                                "needYield" : 0,
                                "saveState" : 1,
                                "restoreState" : 1,
                                "isEOF" : 0,
                                "invalidates" : 0,
                                "keyPattern" : {
                                        "bookingTime" : 1
                                },
                                "indexName" : "bookingTime_1",
                                "isMultiKey" : false,
                                "multiKeyPaths" : {
                                        "bookingTime" : [ ]
                                },
                                "isUnique" : false,
                                "isSparse" : false,
                                "isPartial" : false,
                                "indexVersion" : 2,
                                "direction" : "forward",
                                "indexBounds" : {
                                        "bookingTime" : [
                                                "[new Date(1537518120000), new
Date(1537518120000)]"
                                        ]
                                },
                                "keysExamined" : 1,
                                "seeks" : 1,
                                "dupsTested" : 0,
                                "dupsDropped" : 0,
                                "seenInvalidated" : 0
                        }
                }
        }
```

```
	},
	"serverInfo" : {
		"host" : "Frank-Cf",
		"port" : 27017,
		"version" : "4.0.2",
		"gitVersion" : "fc1573ba18aee42f97a3bb13b67af7d837826b47"
	},
	"ok" : 1
}
```


:

The result above shows that the index update query contains FETCH and IXSCAN phases.

Using driver:

The following example uses the explain() method, bookingTime and bookedBy in the FlightBooking collection:

db.FlightBooking.createIndex({'bookingTime':1}, {'bookedBy':1})

Query with selection filter only by *bookingTime*

db.FlightBooking.createIndex({'bookingTime':1})


```
{
	"queryPlanner" : {
		"plannerVersion" : 1,
		"namespace" : "AirplaneCompany.FlightBooking",
		"indexFilterSet" : false,
		"parsedQuery" : {
			"$and" : [
				{
					"bookingTime" : {
						"$lte" : ISODate("2018-11-07T23:59:59Z")
					}
				},
				{
					"bookingTime" : {
						"$gte" : ISODate("2018-11-07T00:00:00Z")
					}
				}
			]
		},
		"winningPlan" : {
			"stage" : "FETCH",
			"inputStage" : {
				"stage" : "IXSCAN",
				"keyPattern" : {
					"bookingTime" : 1
				},
				"indexName" : "bookingTime_1",
				"isMultiKey" : false,
```

```
                "multiKeyPaths" : {
                        "bookingTime" : [ ]
                },
                "isUnique" : false,
                "isSparse" : false,
                "isPartial" : false,
                "indexVersion" : 2,
                "direction" : "forward",
                "indexBounds" : {
                        "bookingTime" : [
                                "[new Date(1541548800000), new Date(1541635199000)]"
                        ]
                }
        }
    },
    "rejectedPlans" : [ ]
},
"executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 6,
    "executionTimeMillis" : 25,
    "totalKeysExamined" : 6,
    "totalDocsExamined" : 6,
    "executionStages" : {
        "stage" : "FETCH",
        "nReturned" : 6,
        "executionTimeMillisEstimate" : 0,
        "works" : 7,
        "advanced" : 6,
        "needTime" : 0,
        "needYield" : 0,
        "saveState" : 0,
        "restoreState" : 0,
        "isEOF" : 1,
        "invalidates" : 0,
        "docsExamined" : 6,
        "alreadyHasObj" : 0,
        "inputStage" : {
            "stage" : "IXSCAN",
            "nReturned" : 6,
            "executionTimeMillisEstimate" : 0,
            "works" : 7,
            "advanced" : 6,
            "needTime" : 0,
            "needYield" : 0,
            "saveState" : 0,
            "restoreState" : 0,
            "isEOF" : 1,
            "invalidates" : 0,
            "keyPattern" : {
                "bookingTime" : 1
            },
            "indexName" : "bookingTime_1",
            "isMultiKey" : false,
            "multiKeyPaths" : {
```

```
                        "bookingTime" : [ ]
                    },
                    "isUnique" : false,
                    "isSparse" : false,
                    "isPartial" : false,
                    "indexVersion" : 2,
                    "direction" : "forward",
                    "indexBounds" : {
                        "bookingTime" : [
                            "[new Date(1541548800000), new Date(1541635199000)]"
                        ]
                    },
                    "keysExamined" : 6,
                    "seeks" : 1,
                    "dupsTested" : 0,
                    "dupsDropped" : 0,
                    "seenInvalidated" : 0
                }
            }
        },
        "serverInfo" : {
            "host" : "Frank-Cf",
            "port" : 27017,
            "version" : "4.0.2",
            "gitVersion" : "fc1573ba18aee42f97a3bb13b67af7d837826b47"
        },
        "ok" : 1
}
```


Query with selection filter by bookedBy


```
{
        "queryPlanner" : {
            "plannerVersion" : 1,
            "namespace" : "AirplaneCompany.FlightBooking",
            "indexFilterSet" : false,
            "parsedQuery" : {
                "$and" : [
                    {
                        "bookingTime" : {
                            "$lte" : ISODate("2018-11-07T23:59:59Z")
                        }
                    },
                    {
                        "bookingTime" : {
                            "$gte" : ISODate("2018-11-07T00:00:00Z")
                        }
                    }
                ]
            },
            "winningPlan" : {
                "stage" : "COLLSCAN",
                "filter" : {
```

```
                "$and" : [
                    {
                        "bookingTime" : {
                            "$lte" : ISODate("2018-11-07T23:59:59Z")
                        }
                    },
                    {
                        "bookingTime" : {
                            "$gte" : ISODate("2018-11-07T00:00:00Z")
                        }
                    }
                ]
            },
            "direction" : "forward"
        },
        "rejectedPlans" : [ ]
    },
    "executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 7,
        "executionTimeMillis" : 0,
        "totalKeysExamined" : 0,
        "totalDocsExamined" : 1010,
        "executionStages" : {
            "stage" : "COLLSCAN",
            "filter" : {
                "$and" : [
                    {
                        "bookingTime" : {
                            "$lte" : ISODate("2018-11-07T23:59:59Z")
                        }
                    },
                    {
                        "bookingTime" : {
                            "$gte" : ISODate("2018-11-07T00:00:00Z")
                        }
                    }
                ]
            },
            "nReturned" : 7,
            "executionTimeMillisEstimate" : 0,
            "works" : 1012,
            "advanced" : 7,
            "needTime" : 1004,
            "needYield" : 0,
            "saveState" : 7,
            "restoreState" : 7,
            "isEOF" : 1,
            "invalidates" : 0,
            "direction" : "forward",
            "docsExamined" : 1010
        }
    },
    "serverInfo" : {
        "host" : "Frank-Cf",
```

```
            "port" : 27017,
            "version" : "4.0.2",
            "gitVersion" : "fc1573ba18aee42f97a3bb13b67af7d837826b47"
        },
        "ok" : 1
}
```


From the results above, it can be seen that index was not used for query which filtered only by *bookedBy.*

Query with selection filter by bookingTime and bookedBy

db.FlightBooking.createIndex({'bookingTime':1}, 'bookedBy':1)


```
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "AirplaneCompany.FlightBooking",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "$and" : [
                                {
                                        "bookingTime" : {
                                                "$lte" : ISODate("2018-11-07T23:59:59Z")
                                        }
                                },
                                {
                                        "bookingTime" : {
                                                "$gte" : ISODate("2018-11-07T00:00:00Z")
                                        }
                                }
                        ]
                },
                "winningPlan" : {
                        "stage" : "FETCH",
                        "inputStage" : {
                                "stage" : "IXSCAN",
                                "keyPattern" : {
                                        "bookingTime" : 1,
                                        "bookedBy" : 1
                                },
                                "indexName" : "bookingTime_1_bookedBy_1",
                                "isMultiKey" : false,
                                "multiKeyPaths" : {
                                        "bookingTime" : [ ],
                                        "bookedBy" : [ ]
                                },
                                "isUnique" : false,
                                "isSparse" : false,
                                "isPartial" : false,
                                "indexVersion" : 2,
                                "direction" : "forward",
                                "indexBounds" : {
                                        "bookingTime" : [
```

```
                        "[new Date(1541548800000), new Date(1541635199000)]"
                    ],
                    "bookedBy" : [
                        "[MinKey, MaxKey]"
                    ]
                }
            }
        },
        "rejectedPlans" : [ ]
    },
    "executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 5,
        "executionTimeMillis" : 105,
        "totalKeysExamined" : 5,
        "totalDocsExamined" : 5,
        "executionStages" : {
            "stage" : "FETCH",
            "nReturned" : 5,
            "executionTimeMillisEstimate" : 105,
            "works" : 6,
            "advanced" : 5,
            "needTime" : 0,
            "needYield" : 0,
            "saveState" : 1,
            "restoreState" : 1,
            "isEOF" : 1,
            "invalidates" : 0,
            "docsExamined" : 5,
            "alreadyHasObj" : 0,
            "inputStage" : {
                "stage" : "IXSCAN",
                "nReturned" : 5,
                "executionTimeMillisEstimate" : 105,
                "works" : 6,
                "advanced" : 5,
                "needTime" : 0,
                "needYield" : 0,
                "saveState" : 1,
                "restoreState" : 1,
                "isEOF" : 1,
                "invalidates" : 0,
                "keyPattern" : {
                    "bookingTime" : 1,
                    "bookedBy" : 1
                },
                "indexName" : "bookingTime_1_bookedBy_1",
                "isMultiKey" : false,
                "multiKeyPaths" : {
                    "bookingTime" : [ ],
                    "bookedBy" : [ ]
                },
                "isUnique" : false,
                "isSparse" : false,
                "isPartial" : false,
```

```
                        "indexVersion" : 2,
                        "direction" : "forward",
                        "indexBounds" : {
                                "bookingTime" : [
                                        "[new Date(1541548800000), new Date(1541635199000)]"
                                ],
                                "bookedBy" : [
                                        "[MinKey, MaxKey]"
                                ]
                        },
                        "keysExamined" : 5,
                        "seeks" : 1,
                        "dupsTested" : 0,
                        "dupsDropped" : 0,
                        "seenInvalidated" : 0
                }
        }
    },
    "serverInfo" : {
            "host" : "Frank-Cf",
            "port" : 27017,
            "version" : "4.0.2",
            "gitVersion" : "fc1573ba18aee42f97a3bb13b67af7d837826b47"
    },
    "ok" : 1
}
```


Database Profiler

MongoDB has a Performance Advisor that automatically checks for slow queries and suggests new indexes in order to have a better query performance. This collects information about Database Commands executed against a running *mongod* instance. The profiler writes data in the system.profile.collection and to view. The profiling levels available are:

- Level 0 : The profiler is off and does not collect any data (default profiler level)
- Level 1 : The profiler collects data for operations that take longer than the value of **slowms 0 (**by default “slow” operation are considered when they take longer than 100 milliseconds**)**
- Level2 : The profiler includes all operations.

Example

- Set the profiler with level 2

```
db.setProfilingLevel(2)
```

- Run the query on FlightBooking collection

```
db.getCollection('FlightBooking').find({})
```

- Examine system.profiler.collection

```
db.getCollection('system.profile').find({})
```

Having as a result:


```
{
        "op" : "query",
        "ns" : "AirplaneCompany.FlightBooking",
        "command" : {
                        "find" : "FlightBooking",
                        "filter" : {  },
                        "lsid"  :  {"id"  :  UUID("643811cd-373e-43cd-9304-b0036d9df23b")  },
                        "$db" : "AirplaneCompany"
                        },
        "cursorid" : 164201635509,
         "keysExamined" : 0,
         "docsExamined" : 101,
        "numYield" : 0,
        "nreturned" : 101,
        "locks" : {
                        "Global" : {
                                         "acquireCount" : {
                                                 "r" : NumberLong(1)
                                         }
                                },
                        "Database" : {
                                "acquireCount" : {
                                        "r" : NumberLong(1)
                                }
                        },
                        "Collection" : {
                                "acquireCount" : {
                                        "r" : NumberLong(1)
                                }
                        }
                },
        "responseLength" : 42126,
        "protocol" : "op_msg",
        "millis" : 0,
        "planSummary" : "COLLSCAN",
        "execStats" : {
                "stage" : "COLLSCAN",
                "nReturned" : 101,
                "executionTimeMillisEstimate" : 0,
                "works" : 102,
                "advanced" : 101,
                "needTime" : 1,
                "needYield" : 0,
                "saveState" : 1,
                "restoreState" : 0,
                "isEOF" : 0,
                "invalidates" : 0,
                "direction" :"forward",
```

```
                "docsExamined" : 101
        },
        "ts" : ISODate("2018-11-08T21:57:32.028Z"),
        "client" : "127.0.0.1",
        "appName" : "MongoDB Shell",
        "allUsers" : [ ],
         "user" : ""
}
...
```


Setting up the profiler to level 1, inefficient queries can be identified. Then, using explain() method, it can be investigated those queries which was described before and optimised.
This database does not contain lot of documents inside of the collections, apart of the Flightbooking in which it was generated 1000 random queries. All queries run between 0 and 1 milliseconds and this database profiler will not show any slow queries.

How our design is similar to and different a relational system such as SQL?

	MongoDB	**Relational(SQL)**	**Example**
Tables & collections	In MongoDB you do not use tables we use collections instead. Collections are made up of zero or more documents. A document can be safely thought of as a row. A document is made up of one or more fields, which are like columns. Examples of collections we used are 'airlineemployees', 'airports', 'flightbookings and so on'.	If we used a relational system such as a SQL to create our airline company system we would have a number of tables with a number of columns for the airline company system. There are many tables in SQL.	db.Journey.insert({ journeyID: 9765, startingAirport:"Malaga Airport", destinationAirport:"Dubai International DXB", journeyLength: 8068.0 }) *Journey is the collection and it contains fields; journeyID, startingAirport, destinationAirport, and journeyLength.

	Example of a our fields in 'planeFlights' are startingpoint, arrival and departure. There are less collections in MongoDB due to embedded documents.		

Embedded Documents & joins	In our MongoDB system, the developers used one to one relationships with embedded documents. Just because MongoDB does not have joins, the developers use embedded documents when dealing with one to many or many to many relationships. Then stored the data in arrays.	If I was to create our airline company system using SQL, it would not be possible to implement our database with embedded documents as SQL does not support it.
		
		The developer would use two separate two tables(relations) say person and car with the columns.
		
		Each table would have a primary key such as passengerID. To relate these two entities you would have a foreign key field in person with the
			Here is an example of our use of embedded documents:
			db.FlightBooking.insert({
			bookingTime: new Date (2018,8,6,15,00),
			bookedBy: "Abdul Kareem" ,
			flightsUsed: "Emirates Airlines EM2300",
			totalcostofbooking: 5000,
			passengers:[
			{
			name: "Abdul Kareem",
			email: "AdamSmith505@gmail.com",
			number: "07515245454"
			},
			{ name: "Julian London", email: "julianlondon43@gmail.com" number: "0764565435"},

		value of the primary key in car.	
Normalised & denormalization	In MongoDB normalising your data is not used as much as if the developers would implement the airlineCompany in SQL. Normalization in relational database is only feasible under the premise that JOINS between tables are cheap. An alternative to using joins is to denormalise our data. To	In SQL normalization is common as it is optimized for queries and relationships. All rows of one SQL table is stored on one node, so it is possible to join between tables. This would be carried out with a number of join queries.	Here is an example of denormalization in MongoDB. { "_id" : ObjectId("5be49c6d2ad13720b7fb41ac"), "bookingTime" : ISODate("2018-11-08T10:00:00Z"), "bookedBy" : "Drew Mitchell", "flightsUsed" : "BA0107", "totalcostofbooking" : 3000, "passengers" : [{ "name" : "Drew Mitchell", "Email": "bobbykhaleque242@gmail.com", "number" : "+44 751524524"}, {"name" : "Steven Fisher", "email" : "stevenfisher505@gmail.com", "number" : "+44 7562525255"}, { "name" : "Felix Malaga",

	do this we use embedded documents.		"email" : "felixmalage453@gmail.com", "number" : "+44 4714844474"}]} { "_id" : ObjectId("5be49c792ad13720b7fb41ad"), "bookingTime" : ISODate("2018-11-07T10:00:00Z"), "bookedBy" : "Mithcel Jim", "flightsUsed" : "BA0367", "totalcostofbooking" : 3000, "passengers" : [{ "name" : "Drw Mitchell", "email" : "bobbykhaleque242@gmail.com", "number" : "+44 751234524" }, { "name" : "Sten Fisr", "email" : "stever505@gmail.com", "number" : "+44 7562525255" }, { "name" : "Feix Mala", "email" : "felilage453@gmail.com", "number" : "+44 4712344474" }]} { "_id" : ObjectId("5be49c812ad13720b7fb41ae"), "bookingTime" : ISODate("2018-11-07T15:00:00Z"), "bookedBy" : "Abdul Kareem", "flightsUsed" : "BA0117", "totalcostofbooking" : 5000, "passengers" : [{ "name" : "Abdul Kareem", "email" : "AdamSmith505@gmail.com", "number" : "07515245454" }, { "name" : "Julian London", "email" : "julianlondon43@gmail.com", "number" : "0764565435" }, { "name" : "Michael Lucy", "email" : "michaellucy@gmail.com",

			"number" : "07515151221" }]} { "_id" : ObjectId("5be49c8a2ad13720b7fb41af"), "bookingTime" : ISODate("2018-06-07T11:01:00Z"), "bookedBy" : "Frank Senor", "flightsUsed" : "BA0127", "totalcostofbooking" : 4200, "passengers" : [{ "name" : "Ayepapi Senor", "email" : "AS@gmail.com", "number" : "07512344851" }, { "name" : "Sacapuntas Senor", "email" : "SS@gmail.com", "number" : "07542532120" }, { "name" : "Muchograde Senor", "email" : "MS@gmail.com", "number" : "07515134231" }]} { "_id" : ObjectId("5be49c942ad13720b7fb41b0"), "bookingTime" : ISODate("2018-09-07T11:01:00Z"), "bookedBy" : "Sumaya Deen", "flightsUsed" : "BA0127", "totalcostofbooking" : 8000, "passengers" : [{ "name" : "James Deen", "email" : "JD@gmail.com", "number" : "07515234851" }, { "name" : "Mikhdad Deen", "email" : "MD7@gmail.com", "number" : "07542405120" }, { "name" : "Karen Smith", "email" : "KS@gmail.com", "number" : "07578951331" }]} {

"_id" :
ObjectId("5be49c9e2ad13720b7fb41b1"),
"bookingTime" : ISODate("2018-09-12T11:01:00Z"),
"bookedBy" : "Jimmy Goliath",
"flightsUsed" : "BA0117",
"totalcostofbooking" : 9200,
"passengers" : [
{
"name" : "James Goliath",
"email" : "JG5@gmail.com",
"number" : "07234244851"
},
{
"name" : "Phoebe Goliath",
"email" : "PG67@gmail.com",
"number" : "07567805120"
},
{
"name" : "Sadeq Goliath",
"email" :
"sadeqsmith565@gmail.com",
"number" :
"07515156541"
}
]
}
{
"_id" :
ObjectId("5be49ca52ad13720b7fb41b2"),
"bookingTime" : ISODate("2018-05-12T11:01:00Z"),
"bookedBy" : "Jimmothy Dolan",
"flightsUsed" : "BA0107",
"totalcostofbooking" : 3458,
"passengers" : [
{
"name" : "Regina Filange",
"email" : "RF5@gmail.com",
"number" : "07553344851"
},
{
"name" : "Samantha Weller",
"email" : "SW67@gmail.com",
"number" : "07546545120"
},
{
"name" : "Sad Man",
"email" : "SM565@gmail.com",
"number" : "07515187631"}
]}
{

			"_id" : ObjectId("5be49cb02ad13720b7fb41b3"),
			"bookingTime" : ISODate("2018-03-12T01:00:00Z"),
			"bookedBy" : "Johny Devine",
			"flightsUsed" : "BA0367",
			"totalcostofbooking" : 9900,
			"passengers" : [
			{
			"name" : "Eva Rahman",
			"email" : "ER65@gmail.com",
			"number" : "07515424851"},
			{
			"name" : "Shorob Rahman",
			"email" : "SR67@gmail.com",
			"number" : "07542234120"},
			{
			"name" : "Johny Devine",
			"email" : "JD5@gmail.com",
			"number" : "07456151331"}
			]}
			{
			"_id" : ObjectId("5be49cb92ad13720b7fb41b4"),
			"bookingTime" : ISODate("2018-05-11T00:15:00Z"),
			"bookedBy" : "Mikeala Dolo",
			"flightsUsed" : "BA0127",
			"totalcostofbooking" : 7800,
			"passengers" : [
			{
			"name" : "Mikeala Dolo",
			"email" : "Mikaela565@gmail.com",
			"number" : "07515247651"},
			{
			"name" : "Mikhdad Smith",
			"email" : "MikhdadSmith67@gmail.com",
			"number" : "07542404350"},
			{
			"name" : "Sadeg Smith",
			"email" : "sadeqsmith565@gmail.com",
			"number" : "07515153211"}]
			}
			{
			"_id" : ObjectId("5be49cc12ad13720b7fb41b5"),
			"bookingTime" : ISODate("2018-11-07T12:01:00Z"),
			"bookedBy" : "Jamie Smith",

			"flightsUsed" : "BA0117",
			"totalcostofbooking" : 11000,
			"passengers" : [
			{
			"name" : "Khaz Barr",
			"email" : "KB565@gmail.com",
			"number" : "07515244851" },
			{
			"name" : "Rajab Smith",
			"email" : "RS67@gmail.com",
			"number" : "07542405120"},{
			"name" : "Jamie Smith",
			"email" : "JS565@gmail.com",
			"number" : "07515151331"}]}

Foreign keys	MongoDB design does not have foreign keys.	In SQL developers can use both foreign keys and primary keys in their design. For example, in the design we would make a table called Pilot with a primary key called pilotID and we would have a foreign key in planeflights called pilotID.	No example since we used a MongoDB NoSQL design.

Columns & fields	MongoDB documents have one or more fields. A field is an element in which one piece of information is stored.	SQL has columns. A column is a collection of cells aligned vertically in a table. If we used SQL we would have a number of columns and collections for our airlinecompany database.	{
			model: "737-800",
			flyingrange: 5562,
			lengthofservice: 30,
			status: "working",
			seatingcapacity:{
			seat2CLass: 126,
			maximumSeats: 149},
			length: 33.6,
			wingspan: 38.5,
			height: 12.5,
			},
			• Here are fields in our MongoDB design.

Sorting(index)	Indexes in MongoDB function mostly like their RDBMS counterparts. Collections in MongoDB can be indexed, which improves lookup and sorting performance. In MongoDB, if an index is found, every document within a collection must be scanned to select the documents that provide a match to the query statement.	Indexing is possible in MySQL. With MySQL, if an index is not defined, the database will have to scan the entire table to locate all relevant rows.	Here is an example of an index query we used to search the whole collection Using the explain() method with the query: db.FlightBooking.createIndex({ bookingTime: 1}) db.FlightBooking.find({ bookingTime:{$gte: ISODate("2018-11-07T00:00:00.000Z"),$lte: ISODate("2018-11-07T23:59:59.000Z")}}).explain("executionStats") It searches for booking between the two times above.
ObjectID	MongoDB uses ObjectID, which generates a value created by MongoDB's database system.	MySQL works with primary and foreign keys.	{ "_id" : ObjectId("5be49c6d2ad13720b7fb41ac"), "bookingTime" : ISODate("2018-11-08T10:00:00Z"), "bookedBy" : "Drew Mitchell", "flightsUsed" : "BA0107", "totalcostofbooking" : 3000, "passengers" : [{ "name" : "Drew Mitchell", Here is an example of a generated ObjectID in MongoDB.

REFERENCES

1.Docs.mongodb.com. (2018). Explain Results — MongoDB Manual. [online] Available at: https://docs.mongodb.com/manual/reference/explain-results/index.html [Accessed 7 Nov. 2018].

2.Docs.mongodb.com. (2018). db.collection.explain() — MongoDB Manual. [online] Available at: https://docs.mongodb.com/manual/reference/method/db.collection.explain/index.html [Accessed 7 Nov. 2018].

3.Docs.mongodb.com. (2018). Database Profiler — MongoDB Manual. [online] Available at: https://docs.mongodb.com/manual/tutorial/manage-the-database-profiler/index.html [Accessed 7 Nov. 2018].

4.Docs.mongodb.com. (2018). Database Profiler Output — MongoDB Manual. [online] Available at: https://docs.mongodb.com/manual/reference/database-profiler/index.html [Accessed 7 Nov. 2018].

5.Docs.mongodb.com. (2018). Model Relationships Between Documents — MongoDB Manual. [online] Available at: https://docs.mongodb.com/manual/applications/data-models-relationships/ [Accessed 7 Nov. 2018].

6.Blog.panoply.io. (2018). MongoDB vs MySQL: the differences explained. [online] Available at: https://blog.panoply.io/mongodb-and-mysql [Accessed 7 Nov. 2018]

7.Hacker Noon. (2018). MongoDB vs MySQL Comparison: Which Database is Better?. [online] Available at: https://hackernoon.com/mongodb-vs-mysql-comparison-which-database-is-better-e714b699c38b [Accessed 7 Nov. 2018].

8. MongoDB?, H. (2018). How to select a single field in MongoDB?. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/25589113/how-to-select-a-single-field-in-mongodb [Accessed 7 Nov. 2018].

9. Little MongoDB Book(2018) Information about MongoDB[online] Available at Qmplus [Accessed 6 Nov 2018].