DATABASE SYSTEMS

ECS650U/ECS789P - SEMI-STRUCTURED DATA AND ADVANCED DATA MODELLING - 2019/20

GROUP 9
Ali Elzalmy, Zoulfia Negmatovna Hall , Volkan Kunduru

Contents

Section (a): Assumptions

General	
1	All distances are shown in miles
2	All prices are in pound sterling
3	All dates are UST
4	Card payment system, no cash

Drivers	
1	Each driver has history of employment for current company only
2	Each driver can have different cars in the system
3	Each driver has only one employment type, can be changed in the future
4	Any status that defines the car as not roadworthy should not be driven until it is operational again

Cars	
1	More than one car model can be registered with a driver in the system
2	There are four status a car can have: roadworthy, in for service, awaiting repair, written off.
3	Maximum car capacity must not exceed 8 people
4	Price per mile depends on the numbers of seats occupied in a car

Bookings	
1	Bookings are taken over the phone
2	Prices are not fixed and depend on the number of seats occupied and the mileage
3	Contains two client type: corporate and private
4	Accepts two types of bookings: repeat and one time.
5.	Repeat bookings will be carried out by the same driver

Operators	
1	There are eight operators working in shifts
2	The times are recorded for the payment information system

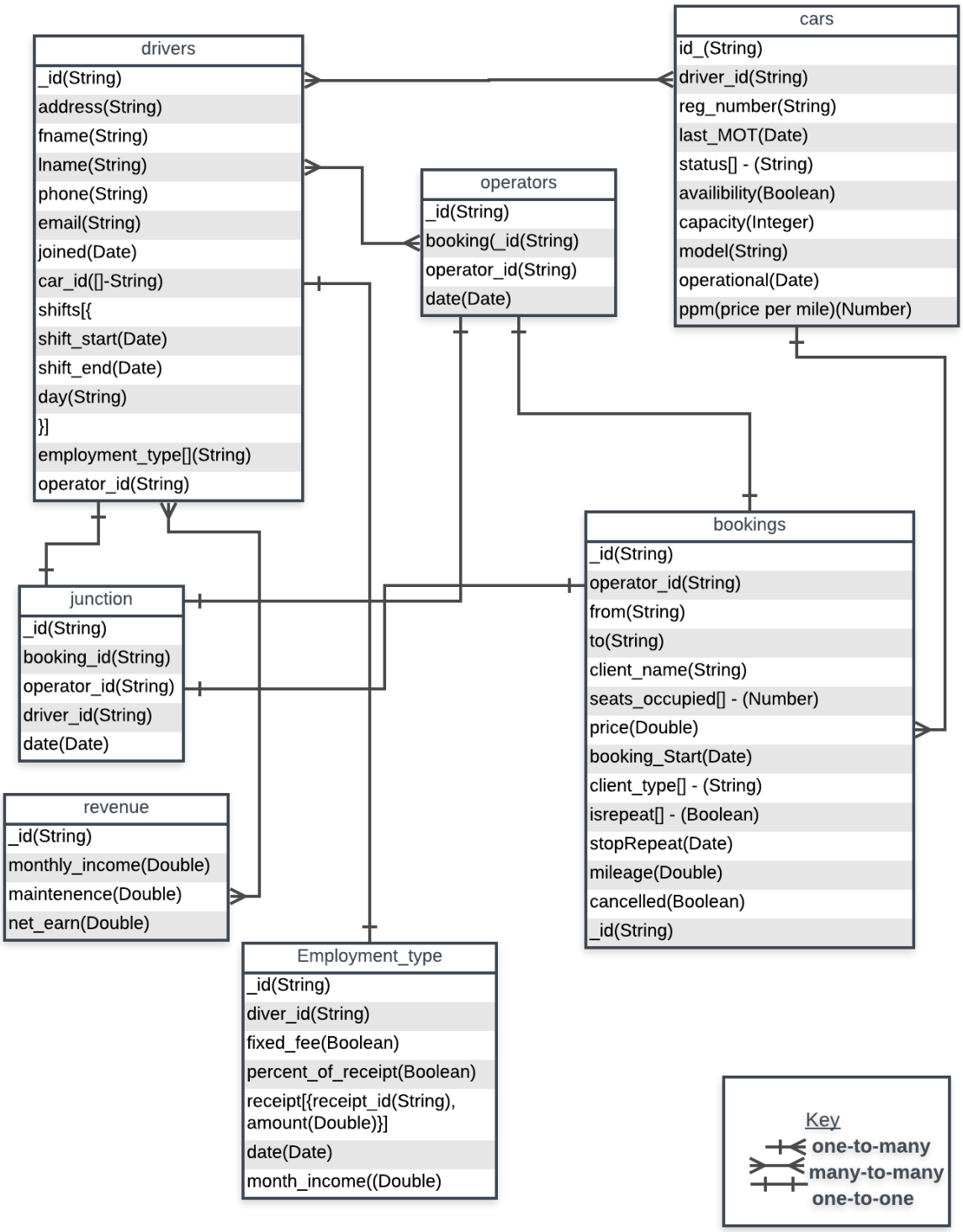
Revenues	
1.	The amount of money earned by drivers is recorded
2	Maintenance fees are deducted from their monthly income to pay operators, lighting, heating and car repairs

Employment_types	
1	There are two types of employment: fixed-fee and percentage_of_receipts.
2	Monthly income of drivers is derived from either the fixed-fee or percentage of receipts earnings.
3	Drivers can change their employment type
4	The total sum of collected receipts are calculated each month

	Junctions
1	Junction collection records data bookings, operators, drivers and date of each booking
2	This collection demonstrates many to many relationship between operators and drivers
3	Allows fast and efficient way of querying collections

RELATIONSHIP	
One car can have many drivers One driver can drive many cars	many to many
One driver has many bookings One booking belongs to one driver	one to many
One driver can have one or more shifts One or more drivers can have the shame shift	many to many
One operator can have one or more shifts One or more operator can have the same shift time	many to many
One operator can have many bookings One booking belongs to one operator	one to many
One employment type belongs to many drivers One employer can have one of two employment types	one to many
One operator can have one or more bookings One booking belongs to one operator	one to many
One driver can have many bookings One booking belongs to one driver	one to many
One car can be used for many bookings One booking use one car	one to many
Revenues have many drivers One driver has many revenues	Many to many
One driver has many operators and vice versa	Many to many

Section (b) UML diagram



Section (c): Design Decision

Collections and attributes:

drivers:

_id: String
address:String
fname: String
lname:String
phone: String
email: String
joined: Date
car_id: [String]
shifts: [{
shift_start: Date
shift_end: Date
day: String
}]
employment_type: String,
operator_id: String

cars

_id:String
driver_id:[String]
reg_num: String
last_MOT: Date
status: String
availability:Boolean
capacity: Number
model: String
operational:Date
ppm: Number
(price per mile)

bookings:

_id:String
operator_id:String
from: String
to: String
client_name: String
seats_occupied: Number
price:Double
booking_start: Date
client_type:String
isRepeat: Boolean
stopRepeat: Date
mileage: Double
cancelled:Boolean
day_of_the_week:[String]

revenues

_id: String
monthly_income: Double
maintenance:Double
net_ernings:Double

employment_types:

_id: String
driver_id:[String]
fixed_fee: Boolean
percentage_of_receipts: Boolean
receipt:[{
receipt_id:String,
amount:Double
}]
date:Date
monthly_income:Double

operators

_id:String
fname:String
lname:String
driver_id: [String]

junctions

_id: String,
booking_id: String,
operator_id: String,
driver_id: String,
date: Date

Drivers

```
const DriverSchema =
        new Schema({
  _id: String,
  car: [CarSchema],
  address: String,
  fname: String,
  lname: String,
  phone: PhoneSchema,
  email: String,
  joined: Date,
  shifts: [ShiftSchema],
  employment_type:String
  operator_id: String
});
```

```
const CarSchema = new Schema({
   _id: String,
   driver_id: [String],
    reg_num: String,
    last_MOT: Date,
   status: String,
   availability: Boolean,
   capacity: Number,
   model: String,
   operational: Date,
   ppm: Number
});
```

```
const PhoneSchema = new Schema({
    work: String,
    mobile: String
  });
```

```
Schema({
   shift_start: Date,
   shift_end: Date,
   day: String,
});
```

The **drivers'** collection stores the information of each driver, giving them a unique identifier, the ID. It stores drivers' details, contact details, employment type, details on their shift times and drivers' employment record (date joined). The driver id is referenced in five other collections: cars, to show who the owner of each car is; bookings, to show which driver is allocated to each booking; revenue, to show which driver earns what, the employment type showing whether a driver is on a fixed-term or percentage-of-receipts basis and operator who assigns jobs to drivers.

Cars

```
const CarSchema = new Schema({
   _id: String,
   driver_id: [String],
    reg_num: String,
    last_MOT: Date,
   status: String,
   availability: Boolean,
   capacity: Number,
   model: String,
   operational: Date,
   ppm: Number
});
```

The **cars** collection stores information about each car, giving them a unique identifier, the ID. It stores their registration number, drivers who drive cars, the date of the last MOT test, the status of the car, and whether it is available or not. The ppm (price per mile) depends on the capacity of the car: if more than 4 seats, ppm increases.

Operators

```
const OperatorSchema = new Schema({
   _id:String,
    fname:String,
    lname:String,
    driver_id: [DriverSchema]
});
  });
```

```
const DriverSchema = new Schema({
   _id: String,
   car: [CarSchema],
   address: String,
   fname: String,
   lname: String,
   phone: PhoneSchema,
   email: String,
   joined: Date,
   shifts: [ShiftSchema],
   employment_type: String,
   operator_id: String
});

});
```

The **operators'** collection stores the details of operators, and drivers' details to whom the job was assigned. The bookings collection contains ids of operators. If we want to find out who was taking bookings for a particular driver the operator_id is used to search this information. As the shifting pattern of drivers correlates to the shifting pattern of operators, there is no need to specify the shifting details for the latter.

Bookings

```
const BookingSchema = new Schema({
  _id:String,
  operator_id: [OperatorSchema],
    from: String,
  to: String,
  client_name: String,
  seats_occupied: Number,
  price:Number,
  booking_start: Date,
  client_type:String,
  isRepeat: Boolean,
  day_of_the_week: [String],
  stopRepeat: Date,
  mileage: Number,
  cancelled:Boolean,
  total_to_pay: Number
```

```
const OperatorSchema = new Schema({
  _id:String,
  fname:String,
  lname:String,
  shifts: [OperatorShiftSchema]
});
```

The **bookings** collection stores information about each booking and gives them a unique identifier, the booking ID. It tells us the operator who did bookings. The operator id has the driver id and the driver id has the car id. From this information drivers and cars details who is going to drive the customer, the car which is going to be used, the number of seats occupied in the car (to determine the price), when the booking starts and how long it is expected to last for, and it will also tell us whether this is a corporate client who will repeat their booking every certain time period. Also, it should say when to stop repeating the booking, and how many miles for calculating the price. The price will be calculated by multiplying the ppm (price per mile) of the car by the number of seats used by the mileage. Operator ids are also registered and whether the booking was canceled. If the booking has not been canceled then the total price for a journey is registered in the system. Booking

contains detailed information about the identity of the driver, what car he has been driven and who was the operator of the date of booking. The driver id must be mentioned as a string to correlate with the driver who was assigned the booking out of the operator's driver_id Array. The reason is that the operator_id can contain many drivers and we need to know which driver was assigned the booking.

Employment Types

Drivers are divided into two categories: those who have a fixed-type employment and those who are percentage-of-receipt based. So, the schema looks like the following:

```
const employment_typeSchema = new Schema({
    _id: String,
    driver_id:[DriverSchema],
    fixed_fee: Boolean,
    percentage_of_receipts: Boolean,
    receipt: [ReceiptSchema],
    monthly_income:Number
});
```

```
const DriverSchema = new Schema({
    _id: String,
    car: [CarSchema],
    address: String,
    fname: String,
    lname: String,
    phone: PhoneSchema,
    email: String,
    joined: Date,
    shifts: [ShiftSchema],
    employment_type: String
    operator_id: String
});
```

```
const ReceiptSchema = new Schema({
    receipt_id:String,
    date:Date,
    amount:Number
``
```

The **employment_types** collection records driver ids, types of employment: fixed-fee or percentage-of-receipt basis. The 'receipt' attribute contains the Receipt schema. It collects the receipts, dates of their collection and generated monthly income. The 'revenues' collection will use this data to calculate the net earnings for drivers.

Revenues

```
const RevenueSchema = new Schema({
    _id:String,
    driver_id: [DriverSchema],
    monthly_income: [employment_typeSchema],
    maintenance:Number,
    net_ernings:Number
});
```

```
const DriverSchema = new Schema({
    _id: String,
    car: [CarSchema],
    address: String,
    fname: String,
    lname: String,
    phone: PhoneSchema,
    email: String,
    joined: Date,
    shifts: [ShiftSchema],
    employment_type: String
   operator_id: String
});
```

```
const employment_typeSchema = new Schema({
   _id: String,
    driver_id:[DriverSchema],
    fixed_fee: Boolean,
    percentage_of_receipts: Boolean,
    receipt: [ReceiptSchema],
    monthly_income:Number
});
```

The **revenue** collection contains drivers' details (DriverSchema), their employment types (employment_typeSchema) and their monthly income. The maintenance fee of 10% is applied to the salaries of drivers to cover costs of lighting, heating, operators' wages and other company necessities. Then the net earnings are calculated. The revenue for each driver depends on their monthly income.

```
const JunctionSchema = new Schema({
        _id: String,
        booking_id: String,
        operator_id: String
        driver_id: String,
        date: Date
});

}
```

The **Junction** collection records data on bookings, operators, drivers, and dates of each booking. This collection shows many to many relationships between operators and drivers. It allows a secure and efficient way of querying collections

Section (d): Inserting documents into collections

In order to fully demonstrate our understanding, we demonstrated that we can insert the queries using both JavaScript and MongoDB query language. We have attached to file named “MongoDBQueries.js” that contains all the MongoDB syntax queries.

The JavaScript language is used to insert the documents into mongo database. To connect our application to the mongo database we used node.js:

```
var url = "mongodb+srv://admin_user:mongodb1234@taxicw-zqlni.gcp.mongodb.net/test?retryWrites=true&w=majority";
new MongoClient(url, { useNewUrlParser: true },{useUnifiedTopology: true}).connect(function(err, db) {
  if (err) throw err;
  var dbo = db.db("taxiCW");
```

1. **The documents were created for insertion.**

```
var car = [new CarModel({
   _id: "C1",
   driver_id:["D1"],
   reg_num: "AA01 0BB",
   last_MOT: new Date("October 21, 2018 10:00:00"),
   status: "roadworthy",
   availability: true,
   capacity: 4,
   model: "Nissan NOTE",
   operational: new Date("October 25, 2018 10:00:00"),
   ppm: 3.60
}), new CarModel({
   _id: "C2",
   driver_id:['D4'],
   reg_num: "AA02 0BB",
   last_MOT: new Date("June 1, 2019 12:00:00"),
   status: "awaiting Repair",
   availability: false,
   capacity: 4,
   model: "Volkswagen",
   operational: new Date("June 23, 2019 08:00:00"),
   ppm: 3.60
})];
```

2. **The 'insertMany' function was used to insert many collections with a single execution:**

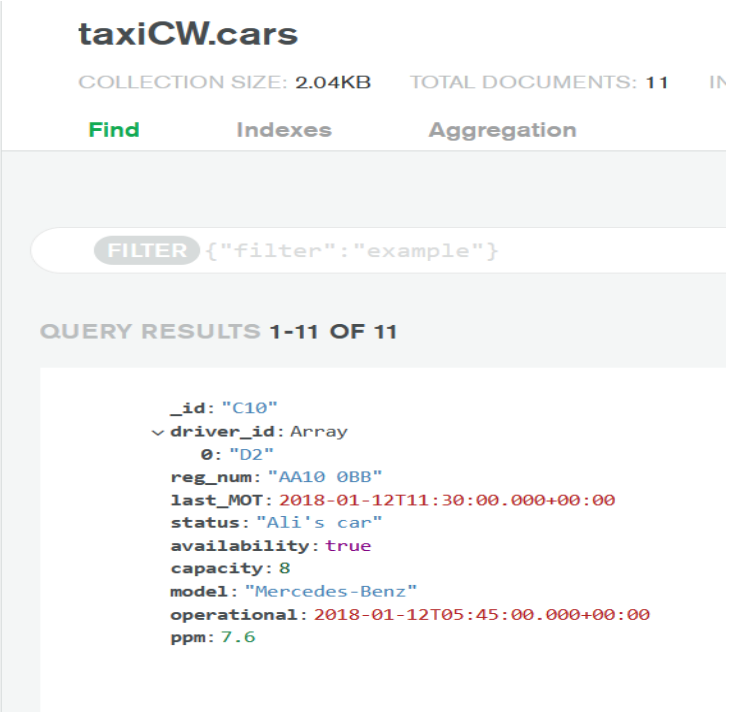
```
dbo.collection("cars").insertMany(car, function(err, res) {
   if (err) throw err;
   console.log("1 document inserted");
 });
```

3.In the terminal the 'node' command was used to execute the 'insertMany' function:

node insertTaxiData.js

1 document inserted

4. The documents were saved in the TaxiCW database on the remote server:



<u>cars collection</u>

```
var car = [new CarModel({
   _id: "C1",
   driver_id:["D1"],
   reg_num: "AA01 0BB",
   last_MOT: new Date("October 21, 2018 10:00:00"),
   status: "roadworthy",
   availability: true,
   capacity: 4,
```

```
    model: "Nissan NOTE",
    operational: new Date("October 25, 2018 10:00:00"),
    ppm: 3.60
}), new CarModel({
    _id: "C2",
    driver_id:['D4'],
    reg_num: "AA02 0BB",
    last_MOT: new Date("June 1, 2019 12:00:00"),
    status: "awaiting Repair",
    availability: false,
    capacity: 4,
    model: "Volkswagen",
    operational: new Date("June 23, 2019 08:00:00"),
    ppm: 3.60
}), new CarModel({
    _id: "C3",
    driver_id:['D6'],
    reg_num: "AA03 0BB",
    last_MOT: new Date("July 15, 2018 12:00:00"),
    status: "in for service",
    availability: false,
    capacity: 8,
    model: "Volkswagen",
    operational: new Date("July 16, 2018 08:00:00"),
    ppm: 5.60
}), new CarModel({
    _id: "C4",
    driver_id:["D1",'D5','D6'],
    reg_num: "AA04 0BB",
    last_MOT: new Date("September 8, 2019 12:00:00"),
    status: "written off",
    availability: false,
    capacity: 4,
    model: "Kia",
    operational: null,
    ppm: 3.60
}), new CarModel({
```

```
    _id: "C5",
    driver_id:["D7"],
    reg_num: "AA05 0BB",
    last_MOT: new Date("August 8, 2018 12:00:00"),
    status: "in Service",
    availability: false,
    capacity: 4,
    model: "Kia",
    operational: new Date("August 9, 2018 11:30:00"),
    ppm: 3.60
}), new CarModel({
    _id: "C6",
    driver_id:["D6"],
    reg_num: "AA06 0BB",
    last_MOT: new Date("March 9, 2018 12:00:00"),
    status: "In Service",
    availability: false,
    capacity: 4,
    model: "Kia",
    operational: new Date("March 9, 2018 02:30:00"),
    ppm: 3.60
}), new CarModel({
    _id: "C7",
    driver_id:["D8"],
    reg_num: "AA07 0BB",
    last_MOT: new Date("October 23, 2018 12:00:00"),
    status: "Roadworthy",
    availability: true,
    capacity: 8,
    model: "Toyota",
    operational: new Date("October 23, 2018 04:30:00"),
    ppm: 5.60
}), new CarModel({
    _id: "C8",
    driver_id:['D4'],
    reg_num: "AA08 0BB",
    last_MOT: new Date("October 23, 2018 12:00:00"),
```

```
    status: "Roadworthy",
    availability: true,
    capacity: 8,
    model: "Toyota",
    operational: new Date("October 23, 2018 04:30:00"),
    ppm: 5.60
}), new CarModel({
    _id: "C9",
    driver_id:["D8"],
    reg_num: "AA09 0BB",
    last_MOT: new Date("April 20, 2018 08:30:00"),
    status: "Roadworthy",
    availability: true,
    capacity: 6,
    model: "Lexus",
    operational: new Date("April 20, 2018 04:28:00"),
    ppm: 7.60
}), new CarModel({
    _id: "C10",
    driver_id:["D2"],
    reg_num: "AA10 0BB",
    last_MOT: new Date("January 12, 2018 11:30:00"),
    status: "Ali's car",
    availability: true,
    capacity: 8,
    model: "Mercedes-Benz",
    operational: new Date("January 12, 2018 05:45:00"),
    ppm: 7.60
}), new CarModel({
    _id: "C11",
    driver_id:["D3"],
    reg_num: "AA11 0BB",
    last_MOT: new Date("January 18, 2018 11:30:00"),
    status: "Volkan's car",
    availability: true,
    capacity: 6,
    model: "BMW",
```

```
    operational: new Date("January 18, 2018 05:45:00"),
    ppm: 6.60
})];
```

drivers collection

```
var driver = [new DriverModel({
    _id: "D1",
    car: [car[0], car[3]],
    address: "10 Purley Rise, Purley, Surrey, CR8 3AU",
    fname: "Zoulfia",
    lname: "Hall",
    phone: new PhoneModel({
        work: "02086688798",
        mobile: "+447745294714"
    }),
    email: "ZoulfiaHall@gmail.com",
    joined: new Date("January 18, 2010 05:45:00"),
    shifts:[new ShiftModel({
            shift_start: "08:30:00",
            shift_end: '17:30:00',
            day: "Monday",

    }), new ShiftModel({
        shift_start: "08:30:00",
        shift_end: "18:30:00",
        day: "Wednesday"

    })],
    employment_type: "percentage_of_receipts",
    operator_id: "O1"
},new DriverModel({
    _id: "D2",
    car: [car[9]],
    address: "14 Down street, London,NW",
    fname: "Ali",
    lname: "Elzalmy",
    phone: new PhoneModel({
```

```
        work: "02086688799",
        mobile: "+447745294715"
    }),
    email: "alielzalmy@gmail.com",
    joined: new Date("May 18, 2000 14:45:00"),
    shifts:[new ShiftModel({
            shift_start: "05:30:00",
            shift_end: "17:30:00",
            day: "Wednesday"

    }), new ShiftModel({
        shift_start: "17:00:00",
        shift_end: "03:30:00",
        day: "Friday"

    })],
    employment_type: "percentage_of_receipts",
    operator_id: "O2"
},new DriverModel({
    _id: "D3",
    car: [car[10]],
    address: "14 Down street, London,NW",
    fname: "Volkan",
    lname: "Kunduru",
    phone: new PhoneModel({
        work: "02086688710",
        mobile: "+447745294716"
    }),
    email: "volkan@gmail.com",
    joined: new Date("July 10, 2011 08:30:00"),
    shifts:[new ShiftModel({
            shift_start: "10:00:00",
            shift_end: "22:00:00",
            day: "Thursday"

    }), new ShiftModel({
        shift_start: "21:00:00",
```

```
        shift_end: "07:00:00",
        day: "Saturday"

    })],
    employment_type: "fixed-fee",
    operator_id: "O3"
},new DriverModel({
    _id: "D4",
    car: [car[1],car[7]],
    address: "10 Downing St, Westminster, London SW1A 2AA",
    fname: "Boris",
    lname: "Johnson",
    phone: new PhoneModel({
        work: "02086688711",
        mobile: "+447745294717"
    }),
    email: "boris@gmail.com",
    joined: new Date("Auguist 15, 2016 08:30:00"),
    shifts:[new ShiftModel({
            shift_start: "07:00:00",
            shift_end: "23:00:00",
            day: "Monday",
            operator_id: ["O7"]
    }), new ShiftModel({
        shift_start: "17:00:00",
        shift_end: "23:59:00",
        day: "Tuesday"

    },new ShiftModel({
        shift_start: "13:00:00",
        shift_end: "02:00:00",
        day: "Wednesday"

    })],
    employment_type: "percentage_of_receipts",
    operator_id: "O4"
},new DriverModel({
```

```
    _id: "D5",
    car: [car[3]],
    address: "14 Down street, London,NW",
    fname: "Sebastian",
    lname: "Hue",
    phone: new PhoneModel({
       work: "02086688712",
       mobile: "+447745294717"
    }),
    email: "sebasian@gmail.com",
    joined: new Date("February 1, 2011 08:30:00"),
    shifts:[new ShiftModel({
          shift_start: "10:00:00",
          shift_end: "22:00:00",
          day: "Thursday"

    }), new ShiftModel({
       shift_start: "21:00:00",
       shift_end: "07:00:00",
       day: "Saturday"

    })],
    employment_type: "fixed-fee",
    operator_id: "O5"
}),new DriverModel({
    _id: "D6",
    car: [car[5],car[2]],
    address: "14 Down street, London,NW",
    fname: "Volkan",
    lname: "Kunduru",
    phone: new PhoneModel({
       work: "02086688710",
       mobile: "+447745294716"
    }),
    email: "volkan@gmail.com",
    joined: new Date("July 10, 2011 08:30:00"),
    shifts:[new ShiftModel({
```

```
            shift_start: "10:00:00",
            shift_end: "22:00:00",
            day: "Thursday"

        }), new ShiftModel({
            shift_start: "21:00:00",
            shift_end: "07:00:00",
            day: "Sunday"

        })],
        employment_type: "fixed-fee",
        operator_id: "O6"
    }),new DriverModel({
        _id: "D7",
        car: [car[4]],
        address: "14 Down street, London,NW",
        fname: "Volkan",
        lname: "Kunduru",
        phone: new PhoneModel({
            work: "02086688710",
            mobile: "+447745294716"
        }),
        email: "volkan@gmail.com",
        joined: new Date("July 10, 2011 08:30:00"),
        shifts:[new ShiftModel({
            shift_start: "08:00:00",
            shift_end: "18:00:00",
            day: "Wednesday"

        }), new ShiftModel({
            shift_start: "21:00:00",
            shift_end: "07:00:00",
            day: "Sunday"

        })],
        employment_type: "percentage_of_receipts",
        operator_id: "O7"
```

```
}),new DriverModel({
_id: "D8",
car: [car[6],car[8]],
address: "14 Down street, London,NW",
fname: "Volkan",
lname: "Kunduru",
phone: new PhoneModel({
    work: "02086688710",
    mobile: "+447745294716"
},
email: "volkan@gmail.com",
joined: new Date("July 10, 2011 08:30:00"),
shifts:[new ShiftModel({
        shift_start: "10:00:00",
        shift_end: "22:00:00",
        day: "Thursday"

}), new ShiftModel({
    shift_start: "21:00:00",
    shift_end: "07:00:00",
    day: "Sunday"

})],
employment_type: "percentage_of_receipts",
operator_id: "O8"
})];
```

<u>Operators collection</u>

```
var operator = [new OperatorModel({
      _id:"O1",
      fname:"Lina",
      lname:"Owen",
      driver_id: [driver[0]]
},new OperatorModel({
      _id:"O2",
      fname:"Laila",
```

```
      lname:"Mehdi",
      driver_id: [driver[1]]

    },new OperatorModel({
      _id:"O3",
      fname:"Tio",
      lname:"Numage",
      driver_id: [driver[2]]
    },new OperatorModel({
      _id:"O4",
      fname:"Roxana",
      lname:"Davidson",
      driver_id: [driver[3]]
},new OperatorModel({
      _id:"O5",
      fname:"Zahra",
      lname:"Ahmed",
      driver_id: [driver[4]]
},new OperatorModel({
      _id:"O6",
      fname:"Tio",
      lname:"Numage",
      driver_id: [driver[5]]
},new OperatorModel({
      _id:"O7",
      fname:"Zanna",
      lname:"Bielecka",
      driver_id: [driver[6]]
   },new OperatorModel({
      _id:"O8",
      fname:"Zizi",
      lname:"Wilsons",
      driver_id: [driver[7]]

   })];
```

employment_type collection

```
var employment_type = [new employment_typeModel({
    _id: "e1",
    driver_id:driver[0],//D1
    fixed_fee: false,
    percentage_of_receipts: true,
    receipt:[new ReceiptModel({
        receipt_id:"r1",
        date: new Date('October 10, 2019 08:30:00'),
        amount: 340.00
    }),new ReceiptModel({
        receipt_id:"r2",
        date: new Date('October 12, 2019 08:30:00'),
        amount: 355.00
    }),new ReceiptModel({
        receipt_id:"r3",
        date: new Date('October 17, 2019 08:30:00'),
        amount: 400.00
    }),new ReceiptModel({
        receipt_id:"r4",
        date: new Date('October 19, 2019 08:30:00'),
        amount: 290.00
    }),new ReceiptModel({
        receipt_id:"r5",
        date: new Date('October 12, 2019 08:30:00'),
        amount: 300.00
    }),new ReceiptModel({
        receipt_id:"r6",
        date: new Date('October 12, 2019 08:30:00'),
        amount: 250.00
    }),new ReceiptModel({
```

```
            receipt_id:"r7",
            date: new Date('October 12, 2019 08:30:00'),
            amount: 300.00
        },new ReceiptModel({
            receipt_id:"r8",
            date: new Date('October 12, 2019 08:30:00'),
            amount: 320.00
        })],
         monthly_income: 2555.00
    },new employment_typeModel({
        _id: "e2",
        driver_id:driver[1],//"D2"
        fixed_fee: false,
        percentage_of_receipts: true,
        receipt:[new ReceiptModel({
            receipt_id:"r9",
            date: new Date('October 7, 2019 08:30:00'),
            amount: 310.00
        },new ReceiptModel({
            receipt_id:"r10",
            date: new Date('October 9, 2019 08:30:00'),
            amount: 335.00
        },new ReceiptModel({
            receipt_id:"r11",
            date: new Date('October 14, 2019 08:30:00'),
            amount: 300.00
        },new ReceiptModel({
            receipt_id:"r12",
            date: new Date('October 16, 2019 08:30:00'),
            amount: 290.00
        },new ReceiptModel({
            receipt_id:"r13",
            date: new Date('October 21, 2019 08:30:00'),
            amount: 400.00
        },new ReceiptModel({
            receipt_id:"r14",
            date: new Date('October 13, 2019 08:30:00'),
```

```
            amount: 250.00
        },new ReceiptModel({
            receipt_id:"r15",
            date: new Date('October 28, 2019 08:30:00'),
            amount: 350.00
        })],
         monthly_income: 2235.00
    },new employment_typeModel({
        _id: "e3",
        driver_id:driver[2],//"D3"
        fixed_fee: true,
        percentage_of_receipts: false,
        receipt:[],
         monthly_income: 2816.00
    },new employment_typeModel({
        _id: "e4",
        driver_id:driver[3],//"D4"
        fixed_fee: false,
        percentage_of_receipts: true,
        receipt:[new ReceiptModel({
            receipt_id:"r1",
            date: new Date('October 1, 2019 08:30:00'),
            amount: 300.00
        },new ReceiptModel({
            receipt_id:"r2",
            date: new Date('October 2, 2019 08:30:00'),
            amount: 350.00
        },new ReceiptModel({
            receipt_id:"r3",
            date: new Date('October 8, 2019 08:30:00'),
            amount: 400.00
        },new ReceiptModel({
            receipt_id:"r4",
            date: new Date('October 9, 2019 08:30:00'),
            amount: 299.00
        },new ReceiptModel({
            receipt_id:"r2",
```

```
            date: new Date('October 15, 2019 08:30:00'),
            amount: 360.00
        }),new ReceiptModel({
            receipt_id:"r2",
            date: new Date('October 16, 2019 08:30:00'),
            amount: 250.00
        }),new ReceiptModel({
            receipt_id:"r2",
            date: new Date('October 22, 2019 08:30:00'),
            amount: 300.00
        }),new ReceiptModel({
            receipt_id:"r2",
            date: new Date('October 29, 2019 08:30:00'),
            amount: 300.00
        }),,new ReceiptModel({
            receipt_id:"r2",
            date: new Date('October 30, 2019 08:30:00'),
            amount: 300.00
        })
    ],
        monthly_income: 2559.00
    }),new employment_typeModel({
        _id: "e5",
        driver_id:driver[4],//"D5"
        fixed_fee: true,
        percentage_of_receipts: false,
        receipt:[],
        monthly_income: 2550.00
    }),new employment_typeModel({
        _id: "e6",
        driver_id:driver[5],//"D6"
        fixed_fee: true,
        percentage_of_receipts: false,
        receipt:[],
        monthly_income: 2405.00
    }),new employment_typeModel({
        _id: "e7",
```

```
driver_id:driver[6],//"D7"
fixed_fee: false,
percentage_of_receipts: true,
receipt:[new ReceiptModel({
    receipt_id:"r17",
    date: new Date('October 2, 2019 08:30:00'),
    amount: 400.00
},new ReceiptModel({
    receipt_id:"r18",
    date: new Date('October 6, 2019 08:30:00'),
    amount: 155.00
},new ReceiptModel({
    receipt_id:"r19",
    date: new Date('October 9, 2019 08:30:00'),
    amount: 400.00
},new ReceiptModel({
    receipt_id:"r20",
    date: new Date('October 13, 2019 08:30:00'),
    amount: 290.00
},new ReceiptModel({
    receipt_id:"r21",
    date: new Date('October 16, 2019 08:30:00'),
    amount: 320.00
},new ReceiptModel({
    receipt_id:"r22",
    date: new Date('October 20, 2019 08:30:00'),
    amount: 250.00
},new ReceiptModel({
    receipt_id:"r23",
    date: new Date('October 23, 2019 08:30:00'),
    amount: 300.00
},new ReceiptModel({
    receipt_id:"r24",
    date: new Date('October 27, 2019 08:30:00'),
    amount: 320.00
},new ReceiptModel({
    receipt_id:"r24",
```

```
            date: new Date('October 30, 2019 08:30:00'),
            amount: 320.00
        })
    ],
     monthly_income: 2355.00
},new employment_typeModel({
    _id: "e8",
    driver_id:driver[7],//"D8"
    fixed_fee: false,
    percentage_of_receipts: true,
    receipt:[new ReceiptModel({
            receipt_id:"r25",
            date: new Date('October 3, 2019 08:30:00'),
            amount: 340.00
    },new ReceiptModel({
            receipt_id:"r26",
            date: new Date('October 6, 2019 08:30:00'),
            amount: 355.00
    },new ReceiptModel({
            receipt_id:"r27",
            date: new Date('October 10, 2019 08:30:00'),
            amount: 200.00
    },new ReceiptModel({
            receipt_id:"r28",
            date: new Date('October 13, 2019 08:30:00'),
            amount: 290.00
    },new ReceiptModel({
            receipt_id:"r29",
            date: new Date('October 17, 2019 08:30:00'),
            amount: 300.00
    },new ReceiptModel({
            receipt_id:"r30",
            date: new Date('October 20, 2019 08:30:00'),
            amount: 290.00
    },new ReceiptModel({
            receipt_id:"r31",
            date: new Date('October 24, 2019 08:30:00'),
```

```
            amount: 300.00
        },new ReceiptModel({
            receipt_id:"r32",
            date: new Date('October 27, 2019 08:30:00'),
            amount: 320.00
        },new ReceiptModel({
            receipt_id:"r32",
            date: new Date('October 31, 2019 08:30:00'),
            amount: 340.00
        })
    ],
        monthly_income: 2735.00
})];
```

<u>revenues collection</u>

```
var revenue = [new RevenueModel({
        _id:"rev1",
        monthly_income: employment_type[0],
        maintenance: 10,
        net_earnings: 2299.50
    },new RevenueModel({
        _id:"rev2",

        monthly_income: employment_type[1],
        maintenance: 10,
        net_earnings: 1011.50
    },new RevenueModel({
        _id:"rev3",

        monthly_income:employment_type[2],
        maintenance: 10,
        net_earnings: 2534.40
    }), new RevenueModel({
        _id:"rev4",

        monthly_income: employment_type[3],
```

```
    maintenance: 10,
    net_earnings: 2303.10
  }),   new RevenueModel({
    _id:"rev5",

    monthly_income: employment_type[4],
    maintenance: 10,
    net_earnings: 2295.00
  }),   new RevenueModel({
    _id:"rev6",

    monthly_income: employment_type[5],
    maintenance: 10,
    net_earnings: 2164.10
  }),   new RevenueModel({
    _id:"rev7",

    monthly_income: employment_type[6],
    maintenance: 10,
    net_earnings: 2119.50
  }),   new RevenueModel({
    _id:"rev8",

    monthly_income: employment_type[7],
    maintenance: 10,
    net_earnings: 2461.50
  })       ];
```

<u>bookings collection</u>

```
var booking = [new BookingModel({
   _id:"B1",
  operator_id:operator[0],
  driver_id: "D1",
  from: "Purley",
  to: "Gatwik airport",
```

```
    client_name: "Tom Wilson",
    seats_occupied: 1,
    price:3.60,
    booking_start: new Date('October 7, 2019 08:30:00'),
    client_type: "private",
    isRepeat: false,
    day_of_the_week:["Wednesday"],
    stopRepeat: null,
    mileage: 20,
    cancelled:false,
    total_to_pay:72.00
},new BookingModel({
     _id:"B2",
    operator_id:operator[1],
    driver_id: "D2",
    from: "Westminster",
    to: "Guilford",
    client_name: "Michael Hall",
    seats_occupied: 3,
    price:6.60,
    booking_start: new Date('October 4, 2019 17:30:00'),
    client_type: "private",
    isRepeat: false,
    day_of_the_week:["Friday"],
    stopRepeat: null,
    mileage: 30,
    cancelled:false,
    total_to_pay:198.00
},new BookingModel({
     _id:"B3",
    operator_id:operator[2],
    driver_id: "D3",
    from: "Croydon",
    to: "Caterham",
    client_name: "Charlie Chaplin",
    seats_occupied: 2,
    price:3.60,
```

```
    booking_start: new Date('October 7, 2019 08:30:00'),
    client_type: "corporate",
    isRepeat: true,
    day_of_the_week:["Thursday"],
    stopRepeat: new Date('July 22, 2020 08:30:00'),
    mileage: 20,
    cancelled:false,
    total_to_pay:3096.00
},new BookingModel({
     _id:"B4",
    operator_id:operator[3],
    driver_id: "D4",
    from: "Wimbledon",
    to: "Richmond",
    client_name: "Luise White",
    seats_occupied: 1,
    price:3.60,
    booking_start: new Date('October 19, 2019 08:30:00'),
    client_type: "private",
    isRepeat: false,
    day_of_the_week:["Tuesday"],
    stopRepeat: null,
    mileage: 6,
    cancelled:false,
    total_to_pay:21.60
},new BookingModel({
     _id:"B5",
    operator_id:operator[4],
    driver_id: "D5",
    from: "Purley",
    to: "Croydon",
    client_name: "Luise White",
    seats_occupied: 6,
    price:7.60,
    booking_start: new Date('October 17, 2019 08:30:00'),
    client_type: "private",
    isRepeat: false,
```

```
    day_of_the_week:["Saturday"],
    stopRepeat: null,
    mileage: 4.5,
    cancelled:false,
    total_to_pay:34.20
},new BookingModel({
     _id:"B6",
    operator_id:operator[5],
    driver_id: "D6",
    from: "Richmond",
    to: "Teddington",
    client_name: "Jimmie Carter",
    seats_occupied: 1,
    price:3.60,
    booking_start: new Date('October 9, 2019 08:30:00'),
    client_type: "private",
    isRepeat: false,
    day_of_the_week:["Sunday"],
    stopRepeat: null,
    mileage: 2,
    cancelled:false,
    total_to_pay:7.20
},new BookingModel({
     _id:"B7",
    operator_id:operator[6],
    driver_id: "D7",
    from: "Guilford",
    to: "Godalming",
    client_name: "Ben Sushe",
    seats_occupied: 8,
    price:3.60,
    booking_start: new Date('October 20, 2019 08:30:00'),
    client_type: "private",
    isRepeat: false,
    day_of_the_week:["Wednesday"],
    stopRepeat: null,
    mileage: 15,
```

```
    cancelled:false,
    total_to_pay:54.00
},new BookingModel({
     _id:"B8",
    operator_id:operator[7],
    driver_id: "D8",
    from: "Mile End",
    to: "Victoria station",
    client_name: "Hue Grant",
    seats_occupied: 1,
    price:3.60,
    booking_start: new Date('October 9, 2019 08:30:00'),
    client_type: "private",
    isRepeat: false,
    day_of_the_week:["Thursday"],
    stopRepeat: null,
    mileage: 10,
    cancelled:true,
    total_to_pay:36
})];
```

junctions collection

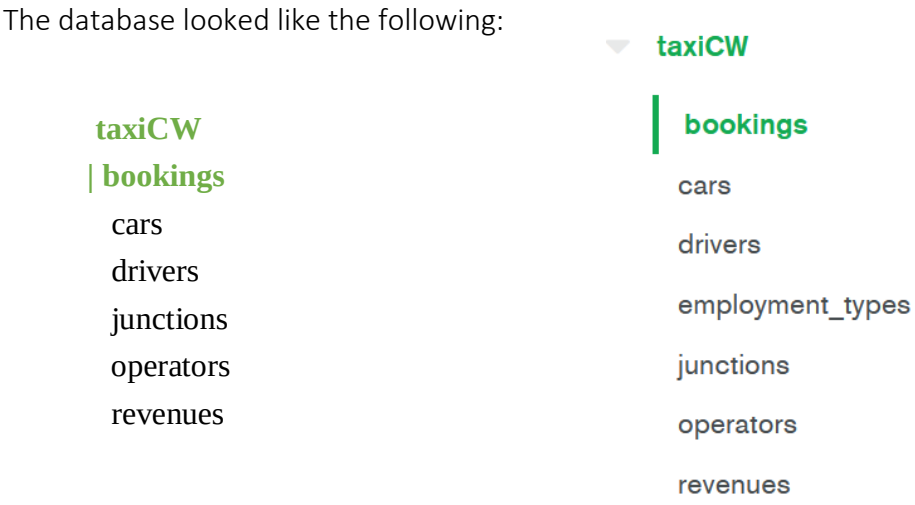
```
var junction = [new JunctionModel({
    _id: "J1",
    booking_id: "B1",
    operator_id: "O1",
    driver_id: "D1",
    date: new Date('October 7, 2019 08:30:00')
}), new JunctionModel({
    _id: "J2",
    booking_id: "B2",
    operator_id: "O2",
    driver_id: "D2",
    date: new Date('October 4, 2019 17:30:00')
}), new JunctionModel({
    _id: "J3",
```

```
    booking_id: "B3",
    operator_id: "O3",
    driver_id: "D3",
    date: new Date('October 4, 2019 17:30:00')
}), new JunctionModel({
    _id: "J4",
    booking_id: "B4",
    operator_id: "O4",
    driver_id: "D4",
    date: new Date('October 19, 2019 08:30:00')
}), new JunctionModel({
    _id: "J5",
    booking_id: "B5",
    operator_id: "O5",
    driver_id: "D5",
    date: new Date('October 17, 2019 08:30:00')
}), new JunctionModel({
    _id: "J6",
    booking_id: "B6",
    operator_id: "O6",
    driver_id: "D6",
    date: new Date('October 9, 2019 08:30:00')
}), new JunctionModel({
    _id: "J7",
    booking_id: "B7",
    operator_id: "O7",
    driver_id: "D7",
    date: new Date('October 20, 2019 08:30:00')
}), new JunctionModel({
    _id: "J8",
    booking_id: "B8",
    operator_id: "O8",
    driver_id: "D8",
    date: new Date('October 9, 2019 08:30:00')
})];
```

The insertion pattern consisting of four steps described above were applied to create six other collections: drivers, operators, employment_types, bookings, junction and revenue.

```
dbo.collection("cars").insertMany(car, function(err, res) {
    if (err) throw err;
    console.log("1 document inserted");
});
dbo.collection("drivers").insertMany(driver, function(err, res) {
    if (err) throw err;
    console.log("1 document inserted");
});

dbo.collection("employment_types").insertMany(employment_type, function(err, res) {
    if (err) throw err;
    console.log("1 document inserted");
});

dbo.collection("revenues").insertMany(revenue, function(err, res) {
    if (err) throw err;
    console.log("1 document inserted");
});

dbo.collection("operators).insertMany(operator, function(err, res) {
    if (err) throw err;
    console.log("1 document inserted");
});

dbo.collection("bookings").insertMany(booking, function(err, res) {
    if (err) throw err;
    console.log("1 document inserted");
});
dbo.collection("junctions").insertMany(junction, function(err, res) {
    if (err) throw err;
    console.log("1 document inserted");
});
```

The database looked like the following:



Section (E): List of Sample Test Data

Each collection contains eight or more records documents

cars


```
_id: "C1"
> driver_id: Array
reg_num: "AA01 0BB"
last_MOT: 2018-10-21T09:00:00.000+00:00
status: "roadworthy"
availability: true
capacity: 4
model: "Nissan NOTE"
operational: 2018-10-25T09:00:00.000+00:00
ppm: 3.6
```

```
_id: "C10"
> driver_id: Array
reg_num: "AA10 0BB"
last_MOT: 2018-01-12T11:30:00.000+00:00
status: "Ali's car"
availability: true
capacity: 8
model: "Mercedes-Benz"
operational: 2018-01-12T05:45:00.000+00:00
ppm: 7.6
```

```
_id: "C11"
> driver_id: Array
reg_num: "AA11 0BB"
last_MOT: 2018-01-18T11:30:00.000+00:00
status: "Volkan's car"
availability: true
capacity: 6
model: "BMW"
operational: 2018-01-18T05:45:00.000+00:00
ppm: 6.6
```

```
_id: "C2"
> driver_id: Array
reg_num: "AA02 0BB"
last_MOT: 2019-06-01T11:00:00.000+00:00
status: "awaiting Repair"
availability: false
capacity: 4
model: "Volkswagen"
operational: 2019-06-23T07:00:00.000+00:00
ppm: 3.6
```

```
_id: "C3"
> driver_id: Array
reg_num: "AA03 0BB"
last_MOT: 2018-07-15T11:00:00.000+00:00
status: "in for service"
availability: false
capacity: 8
model: "Volkswagen"
operational: 2018-07-16T07:00:00.000+00:00
ppm: 5.6
```

```
_id: "C4"
> driver_id: Array
reg_num: "AA04 0BB"
last_MOT: 2019-09-08T11:00:00.000+00:00
status: "written off"
availability: false
capacity: 4
model: "Kia"
operational: null
ppm: 3.6
```

```
_id: "C5"
> driver_id: Array
reg_num: "AA05 0BB"
last_MOT: 2018-08-08T11:00:00.000+00:00
status: "in Service"
availability: false
capacity: 4
model: "Kia"
operational: 2018-08-09T10:30:00.000+00:00
ppm: 3.6
```

```
_id: "C6"
> driver_id: Array
reg_num: "AA06 0BB"
last_MOT: 2018-03-09T12:00:00.000+00:00
status: "In Service"
availability: false
capacity: 4
model: "Kia"
operational: 2018-03-09T02:30:00.000+00:00
ppm: 3.6
```

```
_id: "C7"
> driver_id: Array
reg_num: "AA07 0BB"
last_MOT: 2018-10-23T11:00:00.000+00:00
status: "Roadworthy"
availability: true
capacity: 8
model: "Toyota"
operational: 2018-10-23T03:30:00.000+00:00
ppm: 5.6
```

```
_id: "C8"
> driver_id: Array
reg_num: "AA08 0BB"
last_MOT: 2018-10-23T11:00:00.000+00:00
status: "Roadworthy"
availability: true
capacity: 8
model: "Toyota"
operational: 2018-10-23T03:30:00.000+00:00
ppm: 5.6
```

```
_id: "C9"
> driver_id: Array
reg_num: "AA09 0BB"
last_MOT: 2018-04-20T07:30:00.000+00:00
status: "Roadworthy"
availability: true
capacity: 6
model: "Lexus"
operational: 2018-04-20T03:28:00.000+00:00
ppm: 7.6
```

drivers

```
_id: "D1"
> car: Array
address: "10 Purley Rise, Purley, Surrey, CR8 3AU"
fname: "Zoulfia"
lname: "Hall"
> phone: Object
email: "ZoulfiaHall@gmail.com"
joined: 2010-01-18T05:45:00.000+00:00
> shifts: Array
employment_type: "percentage_of_receipts"
operator_id: "O1"
```

```
_id: "D2"
> car: Array
address: "14 Down street, London,NW"
fname: "Ali"
lname: "Elzalmy"
> phone: Object
email: "alielzalmy@gmail.com"
joined: 2000-05-18T13:45:00.000+00:00
> shifts: Array
employment_type: "percentage_of_receipts"
operator_id: "O2"
```

```
_id: "D3"
> car: Array
address: "14 Down street, London,NW"
fname: "Volkan"
lname: "Kunduru"
> phone: Object
email: "volkan@gmail.com"
joined: 2011-07-10T07:30:00.000+00:00
> shifts: Array
employment_type: "fixed-fee"
operator_id: "O3"
```

```
_id: "D4"
> car: Array
address: "10 Downing St, Westminster, London SW1A 2AA"
fname: "Boris"
lname: "Johnson"
> phone: Object
email: "boris@gmail.com"
joined: 2016-08-15T07:30:00.000+00:00
> shifts: Array
employment_type: "percentage_of_receipts"
operator_id: "O4"
```

```
_id: "D5"
> car: Array
address: "14 Down street, London,NW"
fname: "Sebastian"
lname: "Hue"
> phone: Object
email: "sebasian@gmail.com"
joined: 2011-02-01T08:30:00.000+00:00
> shifts: Array
employment_type: "fixed-fee"
operator_id: "O5"
```

```
_id: "D6"
> car: Array
address: "14 Down street, London,NW"
fname: "Volkan"
lname: "Kunduru"
> phone: Object
email: "volkan@gmail.com"
joined: 2011-07-10T07:30:00.000+00:00
> shifts: Array
employment_type: "fixed-fee"
operator_id: "O6"
```

```
_id: "D7"
> car: Array
address: "14 Down street, London,NW"
fname: "Volkan"
lname: "Kunduru"
> phone: Object
email: "volkan@gmail.com"
joined: 2011-07-10T07:30:00.000+00:00
> shifts: Array
employment_type: "percentage_of_receipts"
operator_id: "O7"
```

```
_id: "D8"
> car: Array
address: "14 Down street, London,NW"
fname: "Volkan"
lname: "Kunduru"
> phone: Object
email: "volkan@gmail.com"
joined: 2011-07-10T07:30:00.000+00:00
> shifts: Array
employment_type: "percentage_of_receipts"
operator_id: "O8"
```

operators

```
_id: "O1"
fname: "Lina"
lname: "Owen"
> driver_id: Array
```

```
_id: "O2"
fname: "Laila"
lname: "Mehdi"
> driver_id: Array
```

```
_id: "O3"
fname: "Tio"
lname: "Numage"
> driver_id: Array
```

```
_id: "O4"
fname: "Roxana"
lname: "Davidson"
> driver_id: Array
```

```
_id: "O5"
fname: "Zahra"
lname: "Ahmed"
> driver_id: Array
```

```
_id: "O6"
fname: "Tio"
lname: "Numage"
> driver_id: Array
```

```
_id: "O7"
fname: "Zanna"
lname: "Bielecka"
> driver_id: Array
```

```
_id: "O8"
fname: "Tio"
lname: "Numage"
> driver_id: Array
```

```
_id: "O9"
fname: "Tio"
lname: "Numage"
> driver_id: Array
```

employment_types

```
_id: "e1"
> driver_id: Array
fixed_fee: false
percentage_of_receipts: true
> receipt: Array
monthly_income: 2555
```

```
_id: "e2"
> driver_id: Array
fixed_fee: false
percentage_of_receipts: true
> receipt: Array
monthly_income: 2235
```

```
_id: "e3"
> driver_id: Array
fixed_fee: true
percentage_of_receipts: false
> receipt: Array
monthly_income: 2816
```

```
_id: "e4"
> driver_id: Array
fixed_fee: false
percentage_of_receipts: true
> receipt: Array
monthly_income: 2559
```

```
_id: "e5"
> driver_id: Array
fixed_fee: true
percentage_of_receipts: false
> receipt: Array
monthly_income: 2550
```

```
_id: "e6"
> driver_id: Array
fixed_fee: true
percentage_of_receipts: false
> receipt: Array
monthly_income: 2405
```

```
_id: "e7"
> driver_id: Array
fixed_fee: false
percentage_of_receipts: true
> receipt: Array
monthly_income: 2355
```

```
_id: "e8"
> driver_id: Array
fixed_fee: false
percentage_of_receipts: true
> receipt: Array
monthly_income: 2735
```

bookings

```
_id: "B1"
> day_of_the_week: Array
> operator_id: Array
from: "Purley"
to: "Gatwik airport"
client_name: "Tom Wilson"
seats_occupied: 1
price: 3.6
booking_start: 2019-10-07T07:30:00.000+00:00
client_type: "private"
isRepeat: false
stopRepeat: null
mileage: 20
cancelled: false
total_to_pay: 72
```

```
_id: "B2"
> day_of_the_week: Array
> operator_id: Array
from: "Westminster"
to: "Guilford"
client_name: "Michael Hall"
seats_occupied: 3
price: 6.6
booking_start: 2019-10-04T16:30:00.000+00:00
client_type: "private"
isRepeat: false
stopRepeat: null
mileage: 30
cancelled: false
total_to_pay: 198
```

```
_id: "B3"
> day_of_the_week: Array
> operator_id: Array
from: "Croydon"
to: "Caterham"
client_name: "Charlie Chaplin"
seats_occupied: 2
price: 3.6
booking_start: 2019-10-07T07:30:00.000+00:00
client_type: "corporate"
isRepeat: true
stopRepeat: 2020-07-22T07:30:00.000+00:00
mileage: 20
cancelled: false
total_to_pay: 3096
```

```
_id: "B4"
> day_of_the_week: Array
> operator_id: Array
from: "Wimbledon"
to: "Richmond"
client_name: "Luise White"
seats_occupied: 1
price: 3.6
booking_start: 2019-10-19T07:30:00.000+00:00
client_type: "private"
isRepeat: false
stopRepeat: null
mileage: 6
cancelled: false
total_to_pay: 21.6
```

```
_id: "B5"
> day_of_the_week: Array
> operator_id: Array
from: "Purley"
to: "Croydon"
client_name: "Luise White"
seats_occupied: 6
price: 7.6
booking_start: 2019-10-17T07:30:00.000+00:00
client_type: "private"
isRepeat: false
stopRepeat: null
mileage: 4.5
cancelled: false
total_to_pay: 34.2
```

```
_id: "B6"
> day_of_the_week: Array
> operator_id: Array
from: "Richmond"
to: "Teddington"
client_name: "Jimmie Carter"
seats_occupied: 1
price: 3.6
booking_start: 2019-10-09T07:30:00.000+00:00
client_type: "private"
isRepeat: false
stopRepeat: null
mileage: 2
cancelled: false
total_to_pay: 7.2
```

```
_id: "B7"
> day_of_the_week: Array
> operator_id: Array
from: "Guilford"
to: "Godalming"
client_name: "Ben Sushe"
seats_occupied: 8
price: 3.6
booking_start: 2019-10-20T07:30:00.000+00:00
client_type: "private"
isRepeat: false
stopRepeat: null
mileage: 15
cancelled: false
total_to_pay: 54
```

```
_id: "B8"
> day_of_the_week: Array
> operator_id: Array
from: "Mile End"
to: "Victoria station"
client_name: "Hue Grant"
seats_occupied: 1
price: 3.6
booking_start: 2019-10-09T07:30:00.000+00:00
client_type: "private"
isRepeat: false
stopRepeat: null
mileage: 10
cancelled: true
total_to_pay: 36
```

revenues

```
_id: "rev1"
> driver_id: Array
> monthly_income: Array
maintenance: 10
net_ernings: 2299.5

_id: "rev2"
> driver_id: Array
> monthly_income: Array
maintenance: 10
net_ernings: 1011.5

_id: "rev3"
> driver_id: Array
> monthly_income: Array
maintenance: 10
net_ernings: 2534.4

_id: "rev4"
> driver_id: Array
> monthly_income: Array
maintenance: 10
net_ernings: 2303.1

_id: "rev5"
> driver_id: Array
> monthly_income: Array
maintenance: 10
net_ernings: 2295

_id: "rev6"
> driver_id: Array
> monthly_income: Array
maintenance: 10
net_ernings: 2164.1

_id: "rev7"
> driver_id: Array
> monthly_income: Array
maintenance: 10
net_ernings: 2119.5

_id: "rev8"
> driver_id: Array
> monthly_income: Array
maintenance: 10
net_ernings: 2461.5
```

junction

```
_id: "J8"
booking_id: "B8"
operator_id: "O8"
driver_id: "D8"
date: 2019-10-09T07:30:00.000+00:00

_id: "J5"
booking_id: "B5"
operator_id: "O5"
driver_id: "D5"
date: 2019-10-17T07:30:00.000+00:00

_id: "J6"
booking_id: "B6"
operator_id: "O6"
driver_id: "D6"
date: 2019-10-09T07:30:00.000+00:00

_id: "J7"
booking_id: "B7"
operator_id: "O7"
driver_id: "D7"
date: 2019-10-20T07:30:00.000+00:00

_id: "J1"
booking_id: "B1"
operator_id: "O1"
driver_id: "D1"
date: 2019-10-07T07:30:00.000+00:00

_id: "J2"
booking_id: "B2"
operator_id: "O2"
driver_id: "D2"
date: 2019-10-04T16:30:00.000+00:00

_id: "J3"
booking_id: "B3"
operator_id: "O3"
driver_id: "D3"
date: 2019-10-04T16:30:00.000+00:00

_id: "J4"
booking_id: "B4"
operator_id: "O4"
driver_id: "D4"
date: 2019-10-19T07:30:00.000+00:00
```

A driver with id D1 drives two cars (C1,C4):

drivers

```
_id: "D1"
car: Array
  0: Object
    driver_id: Array
      0: "D1"
    _id: "C1"
    reg_num: "AA01 0BB"
    last_MOT: 2018-10-21T09:00:00.000+00:00
    status: "roadworthy"
    availability: true
    capacity: 4
    model: "Nissan NOTE"
    operational: 2018-10-25T09:00:00.000+00:00
    ppm: 3.6
  1: Object
    driver_id: Array
      0: "D1"
      1: "D5"
      2: "D6"
    _id: "C4"
    reg_num: "AA04 0BB"
    last_MOT: 2019-09-08T11:00:00.000+00:00
    status: "written off"
    availability: false
    capacity: 4
    model: "Kia"
    operational: null
    ppm: 3.6
address: "10 Purley Rise, Purley, Surrey, CR8 3AU"
fname: "Zoulfia"
lname: "Hall"
phone: Object
  _id: ObjectId("5db0bf042da06b7aa077c241")
  work: "02086688798"
  mobile: "+447745294714"
email: "ZoulfiaHall@gmail.com"
joined: 2010-01-18T05:45:00.000+00:00
shifts: Array
  0: Object
    _id: ObjectId("5db0bf042da06b7aa077c242")
    shift_start: "08:30:00"
    shift_end: "17:30:00"
    day: "Monday"
  1: Object
    _id: ObjectId("5db0bf042da06b7aa077c243")
    shift_start: "08:30:00"
    shift_end: "18:30:00"
    day: "Wednesday"
employment_type: "percentage_of_receipts"
operator_id: "O1"
```

cars

```
_id: "C1"
driver_id: Array
  0: "D1"
reg_num: "AA01 0BB"
last_MOT: 2018-10-21T09:00:00.000+00:00
status: "roadworthy"
availability: true
capacity: 4
model: "Nissan NOTE"
operational: 2018-10-25T09:00:00.000+00:00
ppm: 3.6
```

```
_id: "C4"
driver_id: Array
  0: "D1"
  1: "D5"
  2: "D6"
reg_num: "AA04 0BB"
last_MOT: 2019-09-08T11:00:00.000+00:00
status: "written off"
availability: false
capacity: 4
model: "Kia"
operational: null
ppm: 3.6
```

```
_id: "O1"
fname: "Lina"
lname: "Owen"
driver_id: Array
  0: Object
    _id: "D1"
```

The driver collection has the 'shifts' attribute. The driver (D1) works on Mondays and Wednesdays.

The ' revenues' collection connects to the drivers' details through the 'monthly income' attribute that contains employment type schema. Each receipt is calculated and the monthly income is displayed. This helps to calculate the net earnings for each driver after the maintenance fees of 10% is deducted.

revenues


```
_id: "rev1"
monthly_income: Array
  0: Object
    _id: "e1"
    driver_id: Array
      0: Object
        _id: "D1"
        car: Array
        address: "10 Purley Rise, Purley, Surrey, CR8 3AU"
        fname: "Zoulfia"
        lname: "Hall"
        phone: Object
          _id: ObjectId("5daf5e2d1de61f8674330d5b")
          work: "02086688798"
          mobile: "+447745294714"
        email: "ZoulfiaHall@gmail.com"
        joined: 2010-01-18T05:45:00.000+00:00
        shifts: Array
        employment_type: "percentage_of_receipts"
    fixed_fee: false
    percentage_of_receipts: true
    receipt: Array
      0: Object
        _id: ObjectId("5daf5e2d1de61f8674330da6")
        receipt_id: "r1"
        date: 2019-10-10T07:30:00.000+00:00
        amount: 340
      1: Object
        _id: ObjectId("5daf5e2d1de61f8674330da7")
        receipt_id: "r2"
        date: 2019-10-12T07:30:00.000+00:00
        amount: 355
      2: Object
        _id: ObjectId("5daf5e2d1de61f8674330da8")
        receipt_id: "r3"
        date: 2019-10-17T07:30:00.000+00:00
        amount: 400
      3: Object
        _id: ObjectId("5daf5e2d1de61f8674330da9")
        receipt_id: "r4"
        date: 2019-10-19T07:30:00.000+00:00
        amount: 290
      4: Object
        _id: ObjectId("5daf5e2d1de61f8674330daa")
        receipt_id: "r5"
        date: 2019-10-12T07:30:00.000+00:00
        amount: 300
      5: Object
        _id: ObjectId("5daf5e2d1de61f8674330dab")
        receipt_id: "r6"
        date: 2019-10-12T07:30:00.000+00:00
        amount: 250
      6: Object
        _id: ObjectId("5daf5e2d1de61f8674330dac")
        receipt_id: "r7"
        date: 2019-10-12T07:30:00.000+00:00
        amount: 300
      7: Object
        _id: ObjectId("5daf5e2d1de61f8674330dad")
        receipt_id: "r8"
        date: 2019-10-12T07:30:00.000+00:00
        amount: 320
    monthly_income: 2555
maintenance: 10
net_ernings: 2299.5
```


employment_types


```
_id: "e1"
driver_id: Array
  0: Object
    _id: "D1"
    car: Array
    address: "10 Purley Rise, Purley, Surrey, CR8 3AU"
    fname: "Zoulfia"
    lname: "Hall"
    phone: Object
      _id: ObjectId("5daf112f247c6237f436e471")
      work: "02086688798"
      mobile: "+447745294714"
    email: "ZoulfiaHall@gmail.com"
    joined: 2010-01-18T05:45:00.000+00:00
    shifts: Array
    employment_type: "percentage_of_receipts"
fixed_fee: false
percentage_of_receipts: true
receipt: Array
  0: Object
    _id: ObjectId("5daf112f247c6237f436e53f")
    receipt_id: "r1"
    date: 2019-10-10T07:30:00.000+00:00
    amount: 340
  1: Object
    _id: ObjectId("5daf112f247c6237f436e540")
    receipt_id: "r2"
    date: 2019-10-12T07:30:00.000+00:00
    amount: 355
  2: Object
    _id: ObjectId("5daf112f247c6237f436e541")
    receipt_id: "r3"
    date: 2019-10-17T07:30:00.000+00:00
    amount: 400
  3: Object
    _id: ObjectId("5daf112f247c6237f436e542")
    receipt_id: "r4"
    date: 2019-10-19T07:30:00.000+00:00
    amount: 290
  4: Object
    _id: ObjectId("5daf112f247c6237f436e543")
    receipt_id: "r5"
    date: 2019-10-12T07:30:00.000+00:00
    amount: 300
  5: Object
    _id: ObjectId("5daf112f247c6237f436e544")
    receipt_id: "r6"
    date: 2019-10-12T07:30:00.000+00:00
    amount: 250
  6: Object
    _id: ObjectId("5daf112f247c6237f436e545")
    receipt_id: "r7"
    date: 2019-10-12T07:30:00.000+00:00
    amount: 300
  7: Object
    _id: ObjectId("5daf112f247c6237f436e546")
    receipt_id: "r8"
    date: 2019-10-12T07:30:00.000+00:00
    amount: 320
monthly_income: 2555
```

bookings


```
_id: "B5"
day_of_the_week: Array
    0: "Saturday"
operator_id: Array
    0: Object
        _id: "O5"
        fname: "Zahra"
        lname: "Ahmed"
        driver_id: Array
            0: Object
                _id: "D5"
                car: Array
                    0: Object
                        driver_id: Array
                            0: "D1"
                            1: "D5"
                            2: "D6"
                        _id: "C4"
                        reg_num: "AA04 0BB"
                        last_MOT: 2019-09-08T11:00:00.000+00:00
                        status: "written off"
                        availability: false
                        capacity: 4
                        model: "Kia"
                        operational: null
                        ppm: 3.6
                address: "14 Down street, London,NW"
                fname: "Sebastian"
                lname: "Hue"
                phone: Object
                    _id: ObjectId("5daf4a706fe0ae56a0b05d09")
                    work: "02086688712"
                    mobile: "+447745294717"
                email: "sebasian@gmail.com"
                joined: 2011-02-01T08:30:00.000+00:00
                shifts: Array
                    0: Object
                        operator_id: Array
                            0: "O4"
                        _id: ObjectId("5daf4a706fe0ae56a0b05d0a")
                        shift_start: "10:00:00"
                        shift_end: "22:00:00"
                        day: "Thursday"
                    1: Object
                        operator_id: Array
                            0: "O9"
                        _id: ObjectId("5daf4a706fe0ae56a0b05d0b")
                        shift_start: "21:00:00"
                        shift_end: "07:00:00"
                        day: "Saturday"
                employment_type: "fixed-fee"
from: "Purley"
to: "Croydon"
client_name: "Luise White"
seats_occupied: 6
price: 7.6
booking_start: 2019-10-17T07:30:00.000+00:00
client_type: "private"
isRepeat: false
stopRepeat: null
mileage: 4.5
cancelled: false
total_to_pay: 34.2
```


operators


```
_id: "O5"
fname: "Zahra"
lname: "Ahmed"
driver_id: Array
    0: Object
        _id: "D5"
        car: Array
            0: Object
                driver_id: Array
                    0: "D1"
                    1: "D5"
                    2: "D6"
                _id: "C4"
                reg_num: "AA04 0BB"
                last_MOT: 2019-09-08T11:00:00.000+00:00
                status: "written off"
                availability: false
                capacity: 4
                model: "Kia"
                operational: null
                ppm: 3.6
        address: "14 Down street, London,NW"
        fname: "Sebastian"
        lname: "Hue"
        phone: Object
        email: "sebasian@gmail.com"
        joined: 2011-02-01T08:30:00.000+00:00
        shifts: Array
            0: Object
                operator_id: Array
                _id: ObjectId("5daf4a706fe0ae56a0b05d0a")
                shift_start: "10:00:00"
                shift_end: "22:00:00"
                day: "Thursday"
            1: Object
                operator_id: Array
                _id: ObjectId("5daf4a706fe0ae56a0b05d0b")
                shift_start: "21:00:00"
                shift_end: "07:00:00"
                day: "Saturday"
        employment_type: "fixed-fee"
```


drivers


```
_id: "D5"
car: Array
    0: Object
        driver_id: Array
        _id: "C4"
        reg_num: "AA04 0BB"
        last_MOT: 2019-09-08T11:00:00.000+00:00
        status: "written off"
        availability: false
        capacity: 4
        model: "Kia"
        operational: null
        ppm: 3.6
address: "14 Down street, London,NW"
fname: "Sebastian"
lname: "Hue"
phone: Object
email: "sebasian@gmail.com"
joined: 2011-02-01T08:30:00.000+00:00
shifts: Array
    0: Object
        operator_id: Array
        _id: ObjectId("5daf37db682fb575005f2750")
        shift_start: "10:00:00"
        shift_end: "22:00:00"
        day: "Thursday"
    1: Object
        operator_id: Array
        _id: ObjectId("5daf37db682fb575005f2751")
        shift_start: "21:00:00"
        shift_end: "07:00:00"
        day: "Saturday"
employment_type: "fixed-fee"
```

Section (f): A list of queries to extract information from the system

The document in the **bookings** collection shows how the operator id (O5) connects to the driver id (D5) and through the driver id to the car id (C4). It has a 'cancelled' attributes which sets the value either to true or false. Sometimes customers cancel their bookings. If the 'cancel' attribute is set to 'true', then booking is cancelled.


```
_id: "J8"
booking_id: "B8"
operator_id: "O8"
driver_id: "D8"
date: 2019-10-09T07:30:00.000+00:00
```

```
_id: "O8"
fname: "Zizi"
lname: "Wilsons"
driver_id: Array
  0: Object
    _id: "D8"
    car: Array
    address: "14 Down street, London,NW"
    fname: "Volkan"
    lname: "Kunduru"
    phone: Object
    email: "volkan@gmail.com"
    joined: 2011-07-10T07:30:00.000+00:00
    shifts: Array
    employment_type: "percentage_of_receipts"
```

```
_id: "B8"
day_of_the_week: Array
operator_id: Object
driver_id: "D8"
from: "Mile End"
to: "Victoria station"
client_name: "Hue Grant"
seats_occupied: 1
price: 3.6
booking_start: 2019-10-09T07:30:00.000+00:00
client_type: "private"
isRepeat: false
stopRepeat: null
mileage: 10
cancelled: true
total_to_pay: 36
```

```
_id: "D8"
car: Array
address: "14 Down street, London,NW"
fname: "Volkan"
lname: "Kunduru"
phone: Object
email: "volkan@gmail.com"
joined: 2011-07-10T07:30:00.000+00:00
shifts: Array
employment_type: "percentage_of_receipts"
operator_id: "O8"
```


The junction collection links each booking(B8) with its operator(O8) and driver (D8).

Q1: Find a driver with the id 'D1' that drives two cars (C1,C4):

```
var query2 = {_id:"D1"};
dbo.collection("drivers").find(query2,
{projection: {_id: 0, car: 1}}).toArray((err, result) => {
    result[0]["car"].forEach((item) => {
        console.log("\n" + JSON.stringify(item));
    });
});
```

Output:

{"driver_id":["D1"],"_id":"C1","reg_num":"AA01 0BB","last_MOT":"2018-10-21T09:00:00.000Z","status":"roadworthy","availability":true,"capacity":4,"model":"Nissan NOTE","operational":"2018-10-25T09:00:00.000Z","ppm":3.6}

{"driver_id":["D1","D5","D6"],"_id":"C4","reg_num":"AA04 0BB","last_MOT":"2019-09-08T11:00:00.000Z","status":"written off","availability":false,"capacity":4,"model":"Kia","operational":null,"ppm":3.6}

Q2: Find the driver who drives a BMW and works on Saturdays:

```
var query3 = {$and:[
        {car:   {$elemMatch: {model: "BMW"   }}},
        {shifts:{$elemMatch: {day:   "Saturday"}}}
    ]};
dbo.collection("drivers").findOne(query3, function(err, session) {
    if (err) throw err;
    console.log("\n\n%j", session);
    db.close();
  });
});
```

{
    "_id" : "D3",
    "car" : [
        {
            "driver_id" : [

```
                "D3"
            ],
            "_id" : "C11",
            "reg_num" : "AA11 0BB",
            "last_MOT" : ISODate("2018-01-18T11:30:00Z"),
            "status" : "Volkan's car",
            "availability" : true,
            "capacity" : 6,
            "model" : "BMW",
            "operational" : ISODate("2018-01-18T05:45:00Z"),
            "ppm" : 6.6
        }
    ],
    "address" : "14 Down street, London,NW",
    "fname" : "Volkan",
    "lname" : "Kunduru",
    "phone" : {
        "_id" : ObjectId("5daf566b5688119c605487bf"),
        "work" : "02086688710",
        "mobile" : "+447745294716"
    },
    "email" : "volkan@gmail.com",
    "joined" : ISODate("2011-07-10T07:30:00Z"),
    "shifts" : [
        {
            "_id" : ObjectId("5daf566b5688119c605487c0"),
            "shift_start" : "10:00:00",
            "shift_end" : "22:00:00",
            "day" : "Thursday"
        },
        {
            "_id" : ObjectId("5daf566b5688119c605487c1"),
            "shift_start" : "21:00:00",
            "shift_end" : "07:00:00",
            "day" : "Saturday"
        }
    ],
    "employment_type" : "fixed-fee"
}
```


<u>Q3: The name of the drivers whose net earnings are more than or equal to £2300.00 or less than £2000</u>

```
var query4 = {
    $or: [
```

```
        {net_earnings: {$gte: 2300}},
        {net_earnings: {$lte: 2000}}
    ]
};

var proj4 = {
   projection: {'monthly_income.driver_id.fname': 1,
                'monthly_income.driver_id.lname': 1,
                '_id':                            0}
};

dbo.collection("revenues").find(query4, proj4).toArray( function(err, session) {
   if (err) throw err;
   console.log("\n\n%j", session);
   db.close();
  });
});
```

Or in Mongo shell:

```
> db.revenues.find({  $or: [
... {net_earnings: {$gte: 2300}},
...  {net_earnings: {$lte: 2000}}  ] }).projection({'monthly_income.driver_id.fname': 1,
 'monthly_income.driver_id.lname': 1,  '_id':  0} ).pretty()
{
        "monthly_income" : {
                "driver_id" : {
                        "fname" : "Ali",
                        "lname" : "Elzalmy"
                }
        }
}
{
        "monthly_income" : {
                "driver_id" : {
                        "fname" : "Volkan",
                        "lname" : "Kunduru"
                }
        }
}
{
        "monthly_income" : {
                "driver_id" : {
                        "fname" : "Boris",
                        "lname" : "Johnson"
                }
        }
}
{
        "monthly_income" : {
                "driver_id" : {
                        "fname" : "Volkan",
                        "lname" : "Kunduru"
                }
        }
}
```

<u>Q4. Find the names of the operators and drivers who worked on the day of the booking B1</u>

```
var query5_1 = {
      _id: "B1"
```

```
};

dbo.collection("bookings").findOne(query5_1, function(err, result1) {
   if (err) throw err;
   var days_of_the_week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday", "Sunday"];
   var day_of_the_week = days_of_the_week[result1.booking_start.getDay() - 1];
   console.log("Date of the booking: " + result1.booking_start.toString());

   var query5_2 = {
      shifts: {
         $elemMatch: {
            day: day_of_the_week
         }
      }
   };

   dbo.collection("drivers").find(query5_2, {}).toArray((err, result2) => {result2.forEach(driver
=> {
         if (err) throw err;
         console.log("Driver's Name: ", [driver.fname, driver.lname].join(" "));

         var query5_3 = {
            _id: driver.operator_id
         };
         dbo.collection("operators").findOne(query5_3, (err, result3) => {
            if (err) throw err;
            console.log("Operator's Name: ", [result3.fname, result3.lname].join(" "));
         });
      });
   });

});
```

Output:

```
Date of the booking: Mon Oct 07 2019 08:30:00 GMT+0100 (British Summer Time)
Driver’s Name: Zoulfia Hall
Driver’s Name: Boris Johnson
Operator’s Name: Lina Owen
Operator’s Name: Roxana Davidson
```

Q5. Find the car that had MOT and was written off

```
var query6 = {status:"written off"};
var proj6 = {projection: {last_MOT: 1, status: 1, reg_num: 1}};
dbo.collection("cars").find(query6, proj6).toArray((err, session) => {
   if (err) throw err;
   console.log("\n\n%j", session);
   db.close();
});
```

OR in Mongo shell:

```
Db.cars.find({status:"written off"}).projection({last_MOT: 1, status: 1, reg_num: 1}).pretty()
```

```
{
"_id": "C4",
"reg_num": "AA04 0BB",
"last_MOT": ISODate("2019-09-08T11:00:00Z"),
"status": "written off"
}
```

Q6. Find the booking ID of the booking with the client name that starts with "Ben":

First create an Index.

```
db.bookings.createIndex({client_name: "text"})
{
        "createdCollectionAutomatically": false,
        "numIndexesBefore": 1,
        "numindexesAfter": 2,
        "ok": 1
}
```

Then do the query using the $text index.

```
db.bookings.find({$text: {$search: "Ben"}}).projection({_id: 1}).pretty()
{"_id": "B7"}
```

Q7: Find the client name, price, and journey of the corporate booking(s):

First, get the list of indexes and find the key of the artificial one.

```
> db.bookings.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "taxiCW.bookings"
        },
        {
                "v" : 2,
                "key" : {
                        "_fts" : "text",
                        "_ftsx" : 1
                },
                "name" : "client_name_text",
                "ns" : "taxiCW.bookings",
                "weights" : {
                        "client_name" : 1
                },
                "default_language" : "english",
                "language_override" : "language",
                "textIndexVersion" : 3
        }
]
```

Then, delete the index used in Q6.

db.bookings.dropIndex({"_fts": "text", "_ftsx": 1})

{"nIndexesWas": 2, "ok": 1}

Then, re-create the index used for this question.

db.bookings.createIndex({client_type: "text"})

{

"createdCollectionAutomatically": false,

"numIndexesBefore": 1,

"numindexesAfter": 2,

"ok": 1

}

Then do the query.

db.bookings.find({$text: {$search: “corporate”}}).projection({client_name: 1, _id: 0, from: 1, to: 1, total_to_pay: 1}).pretty()

{

“from”: “Croydon”,

“to”: “Caterham”,

“client_name”: “Charlie Chaplin”,

“total_to_pay”: 3096

}

Q8. Get the cars of the drivers whose monthly income is above 2500 and is on a fixed-fee contract.

```
> db.employment_types.find({fixed_fee: true, percentage_of_receipts: false, monthly_income: {$gte: 2500}})
.projection({driver_id: 1, "driver_id.car": 1, "driver_id.car.reg_num": 1}).pretty()
{
        "_id" : "e3",
        "driver_id" : [
                {
                        "car" : [
                                {
                                        "reg_num" : "AA11 0BB"
                                }
                        ]
                }
        ]
}
{
        "_id" : "e5",
        "driver_id" : [
                {
                        "car" : [
                                {
                                        "reg_num" : "AA04 0BB"
                                }
                        ]
                }
        ]
}
```

We must specify not to show all the attributes of *driver_id* and *driver_id.car* – only the *reg_num* must be shown.

Q9. Get the name of the operator and driver taking the booking_id B3 using the junctions table.

```
var query8 = {booking_id: "B3"};
dbo.collection("junctions").findOne(query8, (err, result) => {
   if (err) throw err;
   dbo.collection("operators").findOne({_id: result.operator_id}, (err, result2) => {
      if (err) throw err;
      console.log("Driver's Name: ", [result2.fname, result2.lname].join(" "));
   });
   dbo.collection("drivers").findOne({_id: result.driver_id}, (err, result2) => {
      if (err) throw err;
      console.log("\nOperator's Name:", [result2.fname, result2.lname].join(" "));
   });
   db.close()
});
```

Output:

```
node query_document.js
```

Driver’s Name: Tio Numage

Operator’s Name: Volkan Kunduru

Q10. Get the name of the driver and client taking one of the bookings handled by the operator O7

```
var query9 = {operator_id: "O7"};
dbo.collection("junctions").findOne(query9, (err, result) => {
   if (err) throw err;
   dbo.collection("drivers").findOne({_id: result.driver_id}, (err, result2) => {
      if (err) throw err;
      console.log("Driver's Name: ", [result2.fname, result2.lname].join(" "));
   });
   dbo.collection("bookings").findOne({_id: result.booking_id}, (err, result2) => {
      if (err) throw err;
      console.log("\nClient's Name: ", result2.client_name);

   });
   db.close();
});
```

Output:

Driver's Name: Volkan Kunduru

Client's Name: Ben Sushe

Section (g): Performance Monitoring Tools – Explain & Profile Commands

db.setProfilingLevel(2)

{"was": 0, "slowms": 100, "sampleRate": 1, "ok": 1}

This is used to make the database start profiling or monitoring our queries.

Then, we can do one query,

db.bookings.find()

for example.

After doing this query, checking the profiler can be done using another query; the data is stored in the collection system.profile – so it can be accessed by doing *db.system.profile.find().* Only the relevant parts of the result will be shown.


```
{
        "op": "query",
        "ns": "taxiCW.bookings",
        "command": {
                "find": "bookings",
                "filter": {},
                "lsid": {
                        "id": UUID("cd7a0047-7c39-40cb-b991-0997df1937b")
                },
                "$db": "taxiCW"
        },
        "keysExamined": 0,
```

```
        "doscExamined": 8,
        "cursorExhausted": true,
        "numYield": 0,
        "nreturned": 8,
        ....
}
```


This shows that it profiled a query (shown in the *op* field). It also demonstrates that the query was performed on the taxiCW.bookings collection, and it was a find command that was used with no filter. It then proceeds to show that there were 8 documents examined, and 8 returned.


```
{...
        "responseLength": 9270,
        "protocol": "op_msg",
        "millis": 0,
        "planSummary": "COLLSCAN",
        "execStats": {
                "stage": "COLLSCAN",
                "nReturned": 8,
                "executionTimeMillisEstimate": 0,
                "works": 10,
                "advanced": 8,
                "needTime": 1,
                "needYield": 0,
                "saveState": 0,
                "restoreState": 0,
                "isEOF": 1,
                "direction": "forward",
                "docsExamined": 8
        },
        "ts": ISODate("2019-10-24T09:43:38.631Z"),
        ...
}
```


The next part of the result shows that the response length was 9270 bytes. Also, the data in *execStats* (execution Statistics) shows us that it took around 0 milliseconds (<1) to complete the query, which is excellent considering it had to search through and return 9270 bytes of

data. The *stage* “COLLSCAN” shows that it was the collection scan execution phase which had 10 work cycles (*works*). The final attribute *ts* shows the date at which the query was executed.

Now, the same query can also be analysed by adding the .explain() command onto the end of the query. We can retrieve the execution stats by using .explain(“executionStats”):

```
> db.bookings.find().explain("executionStats")
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "taxiCW.bookings",
                "indexFilterSet" : false,
                "parsedQuery" : {

                },
                "queryHash" : "8B3D4AB8",
                "planCacheKey" : "8B3D4AB8",
                "winningPlan" : {
                        "stage" : "COLLSCAN",
                        "direction" : "forward"
                },
                "rejectedPlans" : [ ]
        },
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 8,
                "executionTimeMillis" : 0,
                "totalKeysExamined" : 0,
                "totalDocsExamined" : 8,
                "executionStages" : {
                        "stage" : "COLLSCAN",
                        "nReturned" : 8,
                        "executionTimeMillisEstimate" : 0,
                        "works" : 10,
                        "advanced" : 8,
                        "needTime" : 1,
                        "needYield" : 0,
                        "saveState" : 0,
                        "restoreState" : 0,
                        "isEOF" : 1,
                        "direction" : "forward",
                        "docsExamined" : 8
                }
        },
        "serverInfo" : {
                "host" : "DESKTOP-G22R6I5",
                "port" : 27017,
                "version" : "4.2.0",
                "gitVersion" : "a4b751dcf51dd249c5865812b390cfd1c0129c30"
        },
        "ok" : 1
}
```

This is not as detailed as the profiling mentioned earlier, but it can still give some important information such as the following: *namespace* shows that the query was executed on the taxiCW.bookings collection; *parsedQuery* shows there were no filters on the find() command; *serverInfo* shows information about the server such as the port number, the host and the

version of the mongoDB database; also the same *execStats* mentioned in the previous example is present here, giving the exact same output.

If we use these two commands again using a different query, we can show more monitoring data. The query from Q8 in the previous section will be used, which was

```
> db.employment_types.find({fixed_fee: true, percentage_of_receipts: false, monthly_income: {$gte: 2500}})
.projection({driver_id: 1, "driver_id.car": 1, "driver_id.car.reg_num": 1}).pretty()
```

First we will check the system.profile collection:

```
{
        "op" : "query",
        "ns" : "taxiCW.employment_types",
        "command" : {
                "find" : "employment_types",
                "filter" : {
                        "fixed_fee" : true,
                        "percentage_of_receipts" : false,
                        "monthly_income" : {
                                "$gte" : 2500
                        }
                },
                "lsid" : {
                        "id" : UUID("cd7a0047-7c39-40cb-b991-0997df1b937b")
                },
                "$db" : "taxiCW"
        },
        "keysExamined" : 0,
        "docsExamined" : 8,
        "cursorExhausted" : true,
        "numYield" : 0,
        "nreturned" : 2,
```

Right at the start the filter is mentioned in the *filter* attribute which is exactly the same as the parameter I put in the .find() command at the start. Also, we can see that the *ns* (namespace) was different as I was looking through the taxiCW.employment_types table this time. Although it examined 8 documents (*docsExamined)* it only returned the 2 *(nreturned)* which fit the filters.

```
"execStats" : {
        "stage" : "COLLSCAN",
        "filter" : {
                "$and" : [
                        {
                                "fixed_fee" : {
                                        "$eq" : true
                                }
                        },
                        {
                                "percentage_of_receipts" : {
                                        "$eq" : false
                                }
                        },
                        {
                                "monthly_income" : {
                                        "$gte" : 2500
                                }
                        }
                ]
        },
        "nReturned" : 2,
        "executionTimeMillisEstimate" : 0,
        "works" : 10,
        "advanced" : 2,
        "needTime" : 7,
        "needYield" : 0,
        "saveState" : 0,
        "restoreState" : 0,
        "isEOF" : 1,
        "direction" : "forward",
        "docsExamined" : 8
},
```

Later on in the *execStats* attribute it gives us the execution stage (*stage)* which is COLLSCAN again and it also puts the *filter* there again except it is shown in pure query form – instead of simply displaying *key:value* it displays *key: {$eq: value}.* It then details the execution time (0

again!) and it also had 10 works yet again. However, it also had 2 in the *advanced* field. This means that it returned 2 sub-documents, which in this case were the drivers' cars. The *needTime* field showed that the other 7 sub-documents (cars) were not passed back as their drivers did not fit the requirements.

Secondly we will run the .explain("executionStats") command:

```
 "plannerVersion" : 1,
 "namespace" : "taxiCW.employment_types",
 "indexFilterSet" : false,
 "parsedQuery" : {
         "$and" : [
                 {
                         "fixed_fee" : {
                                 "$eq" : true
                         }
                 },
                 {
                         "percentage_of_receipts" : {
                                 "$eq" : false
                         }
                 },
                 {
                         "monthly_income" : {
                                 "$gte" : 2500
                         }
                 }
         ]
 },
```

Again the parsed query is displayed in the *parsedQuery* field.

```
"winningPlan" : {
        "stage" : "COLLSCAN",
        "filter" : {
                "$and" : [
                        {
                                "fixed_fee" : {
                                        "$eq" : true
                                }
                        },
                        {
                                "percentage_of_receipts" : {
                                        "$eq" : false
                                }
                        },
                        {
                                "monthly_income" : {
                                        "$gte" : 2500
                                }
                        }
                ]
        },
        "direction" : "forward"
},
```

And again in the *winningPlan* field. It also shows us the stage COLLSCAN again.

```
 "serverInfo" : {
         "host" : "DESKTOP-G22R6I5",
         "port" : 27017,
         "version" : "4.2.0",
         "gitVersion" : "a4b751dcf51dd249c5865812b390cfd1c0129c30"
 },
 "ok" : 1
```

At the end of the result it shows us the server info again and then the *ok: 1* showing us that the query went ok without any errors

Section (H) Conclusion and summary

In conclusion, we have learnt a lot from this coursework on both how to structure collections and documents and this gave us a first hand insight into MongoDB and how exactly NoSQL database structure differs from SQL database designs. We learnt both how to insert and tried to gain a much deeper understanding into the database design by inserting queries through MongoDB syntax and JavaScript Syntax. The reasoning we did this was because it would give us a more pratical experience and show us that MongoDB has a variety of ways to be recognized and used.

To gain access to our live database:

Enter that through your command terminal (admin_user is the account name).

mongo "mongodb+srv://taxicw-zqlni.gcp.mongodb.net/test" --username admin_user

Password: mongodb1234