Name No: Petra Adu-Tei and Sameen Islam Student No: 140258000 and 140

Coursework 2

MongoDB Design and Implementation

Table of Contents

[Assumptions 3](#_Toc501146160)

[ER Diagram 3](#_Toc501146161)

[Text Description of Collections 4](#_Toc501146162)

[Employee Collection 4](#_Toc501146163)

[Employee Record Collection 4](#_Toc501146164)

[Shift Collection 5](#_Toc501146165)

[Operator Collection 5](#_Toc501146166)

[Driver Collection 5](#_Toc501146167)

[Cars Collection 6](#_Toc501146168)

[Taxi Office Overhead Costs Collection 6](#_Toc501146169)

[Clients Collection 6](#_Toc501146170)

[Payment Collection 6](#_Toc501146171)

[Booking Collection 7](#_Toc501146172)

[MongoDB Queries 7](#_Toc501146173)

[Code: 8](#_Toc501146174)

[Performance Monitoring Tools – Explain() 12](#_Toc501146175)

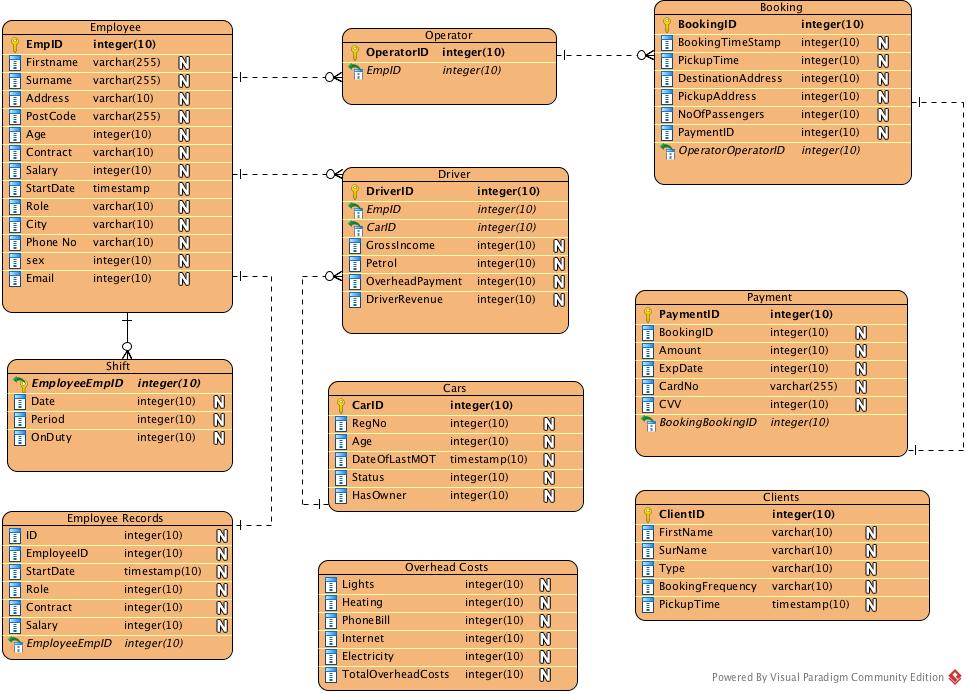
# Assumptions

The assumptions made about the taxi company are documented in the list below:

* Each employee will have a contract which is either a fixed fee on a percentage of revenue earnings
* Employee who are employed on a fixed fee will earn £25000 per annum
* Employee who are employed on a percentage-of-receipts basis will earn 25% of total revenue earnings.
* There are 2 roles for employees either an operator role or driver role
* Employee can work a maximum 2 shifts in a day
* The company has 3 shifts: a morning, afternoon and evening and night shift
* Each driver can only own 1 company car
* For the company overhead costs and car maintenance this will be already deducted from employees’ gross income and will be shown in the revenue the driver.
* The booking clients make will automatically be logged in the booking collection
* Customers will pay for booking via debit/credit card payment only
* Only 1 driver can receive a booking made by an operator
* For a client the booking frequency: monthly, yearly or weekly
* A car ownership by default is set to false until assigned to a user where the field is updated to be true

# ER Diagram

Below shows an ER-Diagram representing the Taxi Company database below.



# Text Description of Collections

The ER-Diagram has 9 collections:

* Employee
* Shift
* Operator
* Driver
* Cars
* Overhead Costs
* Clients
* Payment
* Booking
* Operator

## Employee Collection

The attributes in the collection are as follows:

* id
* Firstname
* Surname
* Address
* PostCode
* Age
* City
* Phone No
* Sex
* Email
* Employment Company Records

Employee has a 1: N relationship with the shift, driver and the operator. To model this object references of employee id, EmpID is put in driver, shift and the operator collections.

## Employee Record Collection

The attributes in the collection are as follows:

* Id
* EmployeeID
* Start date
* Role
* Contract
* Salary

This has a 1:1 Relationship with the employee table, since each employee will directly have an employement record.

## Shift Collection

The attributes for this collection are as follows:

* Date
* Period : This is what shift the driver or operator is on. This can be either morning, afternoon or evening
* OnDuty: This checks if the user is on the shift they are recorded to be on.

The relationship for this collection is a 1: N since an employee can work more than 1 shift in a day. In order to track the employees (driver or operator) we decided to add a reference to the employee ID in our shift collection.

The OnDuty field is an embedded document that consists of the employee id on a particular shift.

## Operator Collection

The attributes for this collection are as follows:

* Id
* EmpId

The relationship for this collection is a 1 to many since an employee could be 1 of the 8 operators. To track which users are employees we decided to use an object reference in the collection.

## Driver Collection

The attributes for this collection are as follows:

* id
* employee\_id
* car\_id
* gross\_income : This documents the total amount the driver earns from the payments of each booking they are allocated.
* driver\_revenue: This is the amount revenuee
* petrol: The amount of petrol used by the driver
* overhead\_payment

A Driver has a 1:N relationship with employees, since an employee can consist of many drivers but a driver is an employee of the company. There is an object reference to the employee company using the employee id: empID.

Driver also has a 1:N relationship with cars, since there are many drivers but a driver can only be assigned one car.

The overhead payment has an indirect relationship with the TaxiOfficeOverhead Collection, it is a derived attribute that will be calculated by dividing the total costs of overpayments

## Cars Collection

The attributes for this collection are as follows:

* id
* RegNo
* Age
* DateOfLastMOT
* Status
* HasOwner

The car collection has a 1: N relationships with drivers, since cars are assigned to many drivers but a driver is only assigned 1 car.

## Taxi Office Overhead Costs Collection

The attributes for this collection are as follows:

* Id
* Lighting
* Heating
* PhoneBill
* Internet
* Electricity
* TotalOverheadCosts

This collection has no direct relationship with any other collection, it simply stores the overhead costs of the taxi company.

## Clients Collection

The attributes for this collection are as follows:

* id
* Firstname
* Surname
* BookingFrequency
* Type
* Pickup

The client doesn’t not have any direct relationship with any other clusters. For the pickup attribute it is an embedded document that consists of the further fields such as address, destination, date, city and postcode

## Payment Collection

The attributes for this collection are as follows:

* id
* booking\_id
* amount
* type
* cardholder\_name
* card\_no
* cvv
* exp

The payment has a 1:1 relationship with booking since each payment is made for each booking made.

## Booking Collection

The attributes for this collection are as follows:

* id
* timestamp
* pickup
* destination
* passenger
* operator\_id

The booking table has a 1:1 relationship with payment each booking will have only 1 payment. To model this relationship, the booking table has a reference of the payment collection ID.

The booking table also has a 1:N relationship with operator, since an operator can make many bookings but a booking can only be made by 1 operator.

The pickup attribute is also an embedded document it includes the following fields: address, destination, date, city and postcode.

The destination attribute is also an embedded document, it includes the following fields: address city and postcode.

# MongoDB Queries

**Query 1: How many clients have a monthly booking?**

**Code**

db.Client.find**({**BookingFrequency**:** "Monthly"**})**

**Output**

"\_id" **:** ObjectId**(**"5a32dafda4b0091db379175f"**),** "Firstname" **:** "Kate"**,** "Surname" **:** "Middleton"**,** "BookingFrequency" **:** "Monthly"**,** "Type" **:** "Corporate"**,** "Pickup" **:** **{** "Address" **:** "34 Strathdon Lane"**,** "PostCode" **:** "SW11 5FE"**,** "City" **:** "London"**,** "Date" **:** ISODate**(**"2017-12-14T20:11:41.009Z"**)** **}** **}**

**{** "\_id" **:** ObjectId**(**"5a32dfdc8a5a6d987212a018"**),** "Firstname" **:** "Kim"**,** "Surname" **:** "Pitti"**,** "BookingFrequency" **:** "Monthly"**,** "Type" **:** "Corporate"**,** "Pickup" **:** **{** "Address" **:** "34 Strathdon Lane"**,** "PostCode" **:** "SW11 5FE"**,** "City" **:** "London"**,** "Date" **:** ISODate**(**"2017-12-14T20:32:28.844Z"**)** **}** **}**

**{** "\_id" **:** ObjectId**(**"5a32e0a88a5a6d987212a019"**),** "Firstname" **:** "Kim"**,** "Surname" **:** "Karda"**,** "BookingFrequency" **:** "Monthly"**,** "Type" **:** "Corporate"**,** "Pickup" **:** **{** "Address" **:** "34 Strathdon Lane"**,** "PostCode" **:** "SW11 5FE"**,** "City" **:** "London"**,** "Date" **:** ISODate**(**"2017-12-14T20:35:51.983Z"**)** **}** **}**

**Query 2: Find the Female employees that are born after 1990 by DOB**

|  |
| --- |
| **Code**: |
| db.employees.find( { $and: [ { dob: { $gt: new Date("1990") } }, { sex: { $eq: "F" } } ] }, { first\_name: 1, last\_name: 1, email: 1, phone: 1 } ).sort( { dob: 1 } ); |
|  |
| **Output**  { "\_id" : ObjectId("5a342c1e6599838054517599"), "first\_name" : "Kate", "last\_name" : "Michelin", "email" : "Kate@hotmail.co.uk", "phone" : "0743275922006" }  { "\_id" : ObjectId("5a342c1e6599838054517597"), "first\_name" : "Isabella", "last\_name" : "Camille", "email" : "izzy@hotmail.co.uk", "phone" : "0743275922006" } |

**Query 3: Find the female employees born after 1989 with their age rounded down and contact details shown, sorted by ascending age**

**Code**:

**Query 4: How many employees work on a morning and evening shift**

|  |
| --- |
| db.employees.aggregate( [ { $match: { $and: [ { dob: { $gt: new Date("1990") } }, { sex: { $eq: "F" } } ] } }, { $sort: { "dob": -1 } }, { $project: { \_id: 0, first\_name: 1, last\_name: 1, "age": { $floor : { $divide: [ { $subtract: [ new Date(), "$dob" ] }, (1000 \* 86400 \* 365 ) ] } }, email: 1, phone: 1 } } ] );  **Output:**  { "first\_name" : "Isabella", "last\_name" : "Camille", "email" : "izzy@hotmail.co.uk", "phone" : "0743275922006", "age" : 24 }  { "first\_name" : "Kate", "last\_name" : "Michelin", "email" : "Kate@hotmail.co.uk", "phone" : "0743275922006", "age" : 27 } |

# **Code:**

var morning = db.shifts.find({period :"morning"}).count()

var afternoon = db.shifts.find({period :"afternoon"}).count()

var total = morning + afternoon

total

**Output:**

4

**Query 5: No. of drivers working for each shift period in a given month.**

|  |
| --- |
| Code: |
|  | db.shifts.aggregate( [ { $unwind: "$on\_duty" }, { $lookup: { from: "drivers" , localField: "on\_duty.employee\_id", foreignField: "employee\_id", as: "driver\_docs"} }, { $project: { date: 1, period: 1, on\_duty: 1, "driver\_docs": { $filter: { input: "$driver\_docs", as: "driver\_doc", cond: { $eq: [ "$$driver\_doc.employee\_id", "on\_duty.employee\_id" ] } } } } } ] ); |

Output:

2

**Query 6: Calculate the total taxi company overhead costs**

Code: db.overheadCosts.aggregate( [ { $project: { total: { $sum: [ "$lighting", "$heating", "$fuel"] } } } ] )

Output

{

"\_id" : ObjectId("5a342863dc4597cc7769b9ed"),

"total" : 5110.0

}

**Query 7: Get the number of duplicate employees by their address**

|  |
| --- |
| **Code:** |
|  | db.employees.aggregate( [ |
|  | { $group: { |
|  | \_id: { email: "$email" }, |
|  | uniqueEmails: { $addToSet: "$\_id" }, |
|  | count: { $sum: 1 } |
|  | } |
|  | }, |
|  | { $match: { |
|  | count: { "$gt": 1 } |
|  | } |
|  | }, |
|  | { |
|  | $sort: { |
|  | count: -1 |
|  | } |
|  | } |
|  | ]) |

Output:

{

"\_id" : {

"email" : "izzy@hotmail.co.uk"

},

"uniqueEmails" : [

ObjectId("5a34366e1056d48943c216cf"),

ObjectId("5a317acdcbf7aee91dbc50d5")

],

"count" : 2.0

}

{

"\_id" : {

"email" : "john\_r@hotmail.com"

},

"uniqueEmails" : [

ObjectId("5a34366e1056d48943c216ce"),

ObjectId("5a317acdcbf7aee91dbc50d4")

],

"count" : 2.0

}

{

"\_id" : {

"email" : "skye@gmail.co.uk"

},

"uniqueEmails" : [

ObjectId("5a34366e1056d48943c216cd"),

ObjectId("5a317acdcbf7aee91dbc50d3")

],

"count" : 2.0

}

{

"\_id" : {

"email" : "marta@gmail.com"

},

"uniqueEmails" : [

ObjectId("5a34366e1056d48943c216cc"),

ObjectId("5a317acdcbf7aee91dbc50d2")

],

"count" : 2.0

}

**Query 8: How many cars are age 5 and over?**

Code:

db.Car.find**({**Age**:{**$gt**:**5**}})**

Output:

{ "\_id" : ObjectId("5a31dedee57c9f977808718c"), "RegNo" : "ADYH4516", "Age" : 11, "DateOfLastMOT" : ISODate("1970-01-01T00:00:00Z"), "Status" : "In For Service", "HasOwner" : false }

{ "\_id" : ObjectId("5a31dedee57c9f977808718e"), "RegNo" : "PDFO4516", "Age" : 7, "DateOfLastMOT" : ISODate("2017-10-22T00:00:00Z"), "Status" : "Roadworthy", "HasOwner" : false }

{ "\_id" : ObjectId("5a31dedee57c9f9778087190"), "RegNo" : "JK78G816", "Age" : 20, "DateOfLastMOT" : ISODate("2017-08-22T00:00:00Z"), "Status" : "Written Off", "HasOwner" : false }

**Query 9: How many clients have a booking frequency that is yearly ?**

Code: db.Client.find({BookingFrequency:"Yearly"})

Output:

{ "\_id" : ObjectId("5a341ec0724b1970c3c7c31a"), "Firstname" : "Kim", "Surname" : "Pitti", "BookingFrequency" : "Yearly", "Type" : "Corporate", "Pickup" : { "Address" : "34 Strathdon Lane", "PostCode" : "SW11 5FE", "City" : "London", "Date" : ISODate("2017-12-14T20:32:28.844Z") } }

{ "\_id" : ObjectId("5a341ec0724b1970c3c7c31c"), "Firstname" : "Tim", "Surname" : "Push", "BookingFrequency" : "Yearly", "Type" : "Corporate", "Pickup" : { "Address" : "34 Kimchoo Lane", "PostCode" : "SW14 5FE", "City" : "London", "Date" : ISODate("2017-12-14T20:35:51.983Z") } }

{ "\_id" : ObjectId("5a341ec0724b1970c3c7c31d"), "Firstname" : "Helen", "Surname" : "Kent", "BookingFrequency" : "Yearly", "Type" : "Corporate", "Pickup" : { "Address" : "34 Kimchoo Lane", "PostCode" : "SW14 5FE", "City" : "London", "Date" : ISODate("2017-12-14T20:35:51.983Z") } }

{ "\_id" : ObjectId("5a341ec0724b1970c3c7c31e"), "Firstname" : "Taylor", "Surname" : "Swift", "BookingFrequency" : "Yearly", "Type" : "Corporate", "Pickup" : { "Address" : "34 Strathdon Lane", "PostCode" : "SW11 5FE", "City" : "London", "Date" : ISODate("2017-12-14T20:11:31.009Z") } }

**Query 10: How many users are corporate users?**

Code: db.Client.find({Type:"Corporate"})

Output:

{ "\_id" : ObjectId("5a341ec0724b1970c3c7c319"), "Firstname" : "Kate", "Surname" : "Middleton", "BookingFrequency" : "Monthly", "Type" : "Corporate", "Pickup" : { "Address" : "34 Strathdon Lane", "PostCode" : "SW11 5FE", "City" : "London", "Date" : ISODate("2017-12-14T20:11:41.009Z") } }

{ "\_id" : ObjectId("5a341ec0724b1970c3c7c31a"), "Firstname" : "Kim", "Surname" : "Pitti", "BookingFrequency" : "Yearly", "Type" : "Corporate", "Pickup" : { "Address" : "34 Strathdon Lane", "PostCode" : "SW11 5FE", "City" : "London", "Date" : ISODate("2017-12-14T20:32:28.844Z") } }

{ "\_id" : ObjectId("5a341ec0724b1970c3c7c31b"), "Firstname" : "Kim", "Surname" : "Karda", "BookingFrequency" : "Monthly", "Type" : "Corporate", "Pickup" : { "Address" : "34 Strathdon Lane", "PostCode" : "SW11 5FE", "City" : "London", "Date" : ISODate("2017-12-14T20:35:51.983Z") } }

{ "\_id" : ObjectId("5a341ec0724b1970c3c7c31c"), "Firstname" : "Tim", "Surname" : "Push", "BookingFrequency" : "Yearly", "Type" : "Corporate", "Pickup" : { "Address" : "34 Kimchoo Lane", "PostCode" : "SW14 5FE", "City" : "London", "Date" : ISODate("2017-12-14T20:35:51.983Z") } }

{ "\_id" : ObjectId("5a341ec0724b1970c3c7c31d"), "Firstname" : "Helen", "Surname" : "Kent", "BookingFrequency" : "Yearly", "Type" : "Corporate", "Pickup" : { "Address" : "34 Kimchoo Lane", "PostCode" : "SW14 5FE", "City" : "London", "Date" : ISODate("2017-12-14T20:35:51.983Z") } }

{ "\_id" : ObjectId("5a341ec0724b1970c3c7c31e"), "Firstname" : "Taylor", "Surname" : "Swift", "BookingFrequency" : "Yearly", "Type" : "Corporate", "Pickup" : { "Address" : "34 Strathdon Lane", "PostCode" : "SW11 5FE", "City" : "London", "Date" : ISODate("2017-12-14T20:11:31.009Z") } }

# Performance Monitoring Tools – Explain()

db.employees.find({ city: "London" }).pretty().explain('executionStats')

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 8,

"executionTimeMillis" : 0,

"totalKeysExamined" : 0,

"totalDocsExamined" : 11,

"executionStages" : {

"stage" : "COLLSCAN",

"filter" : {

"city" : {

"$eq" : "London"

}

},

"nReturned" : 8,

"executionTimeMillisEstimate" : 0,

"works" : 13,

"advanced" : 8,

"needTime" : 4,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"invalidates" : 0,

"direction" : "forward",

"docsExamined" : 11

}

},

}

In this query, we pass in 'executionStats' argument to explain method, which produces various statistics on the execution of

the query such as if its execution was successful and the number of documents returned. By inspecting 'totalDocsExamined', we see

that all 11 documents were traversed - this is further backed by the presence of 'COLLSCAN' under 'executionStages' which, as stated

previously, implies that the collection is not indexed.