LIGHT LIBERTY LEARNING

TASK 2

Sophie Fidan
21068639

UFCFU3-15-3
Advanced

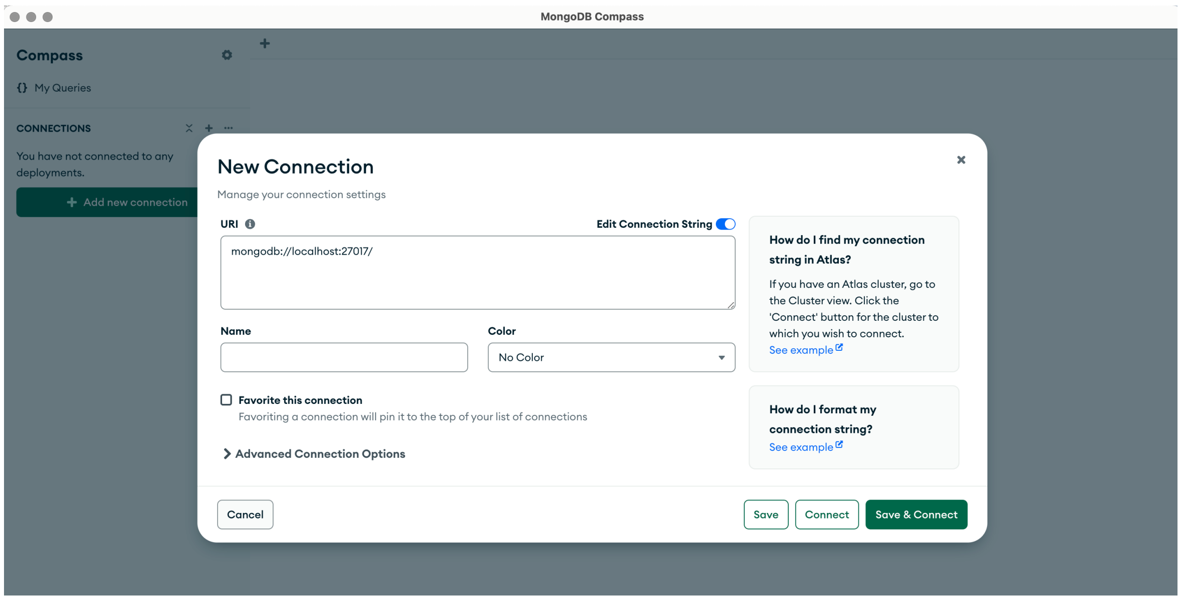
Databases

TABLE OF CONTENTS

MONGODB SETUP

MongoDB setup using MongoDB Compass, and the data was imported straight from the CSV file.

CONNECTING LOCALHOST



New Connection

Manage your connection settings

URI: mongodb://localhost:27017/

Edit Connection String: on

Name:

Color: No Color

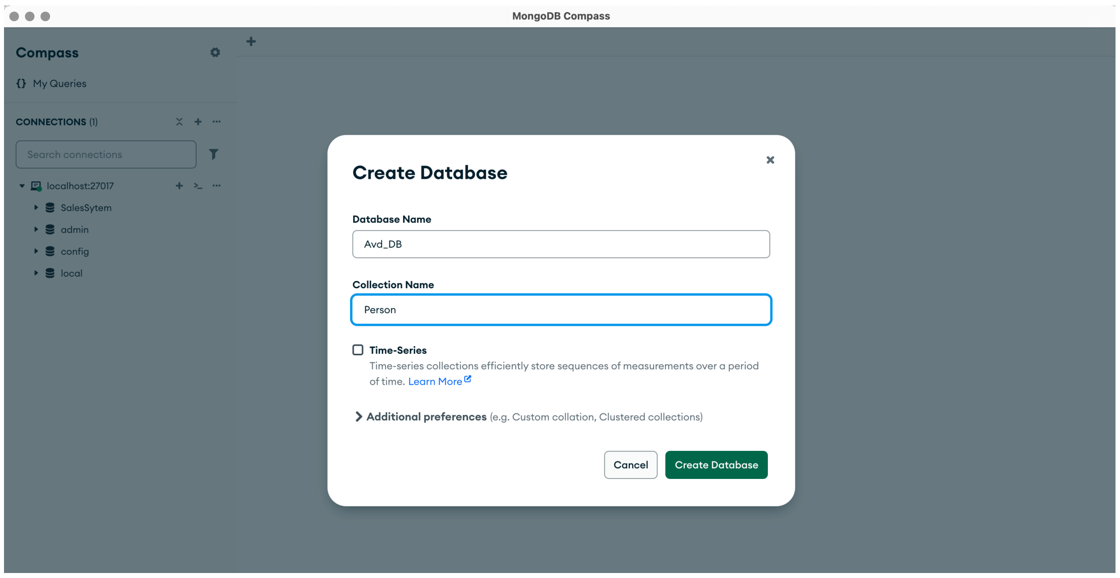
Favorite this connection
Favoriting a connection will pin it to the top of your list of connections

How do I find my connection string in Atlas?
If you have an Atlas cluster, go to the Cluster view. Click the 'Connect' button for the cluster to which you wish to connect.
See example

How do I format my connection string?
See example

You have not connected to any deployments.

CREATING A NEW DATABASE AND COLLECTION



Create Database

Database Name: Avd_DB

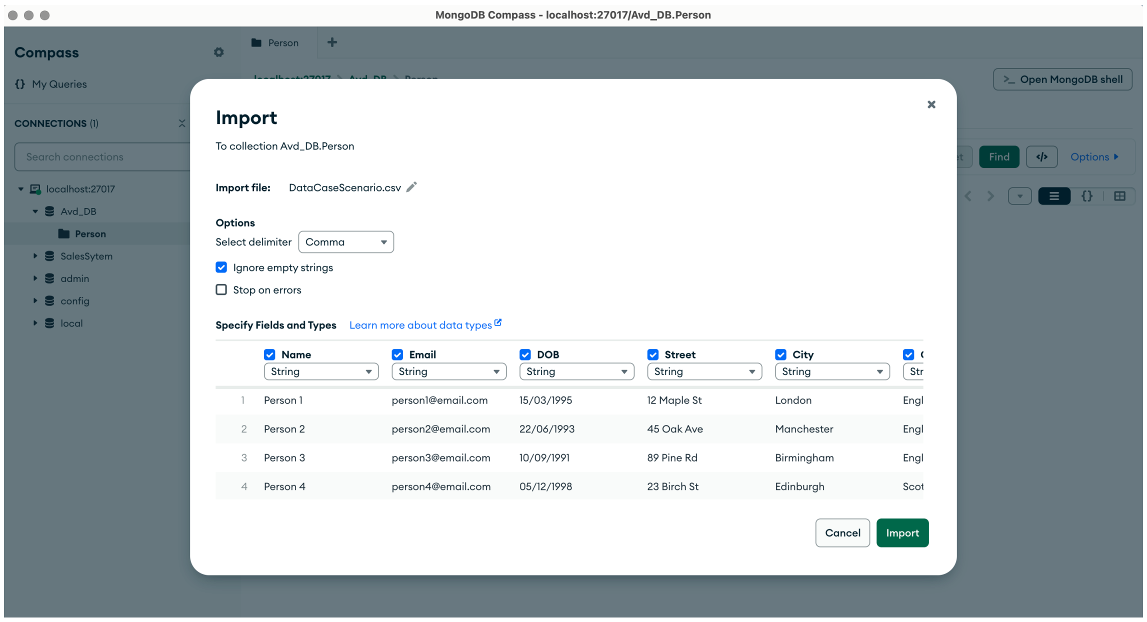
Collection Name: Person

Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. Learn More

Additional preferences (e.g. Custom collation, Clustered collections)

Note: In a NoSQL model, a single collection (table) can be used to store all data in a denormalised format to simplify data retrieval and reduce complexity.

IMPORTING DATA FROM CSV FILE



Import

To collection Avd_DB.Person

Import file: DataCaseScenario.csv

Options

Select delimiter: Comma

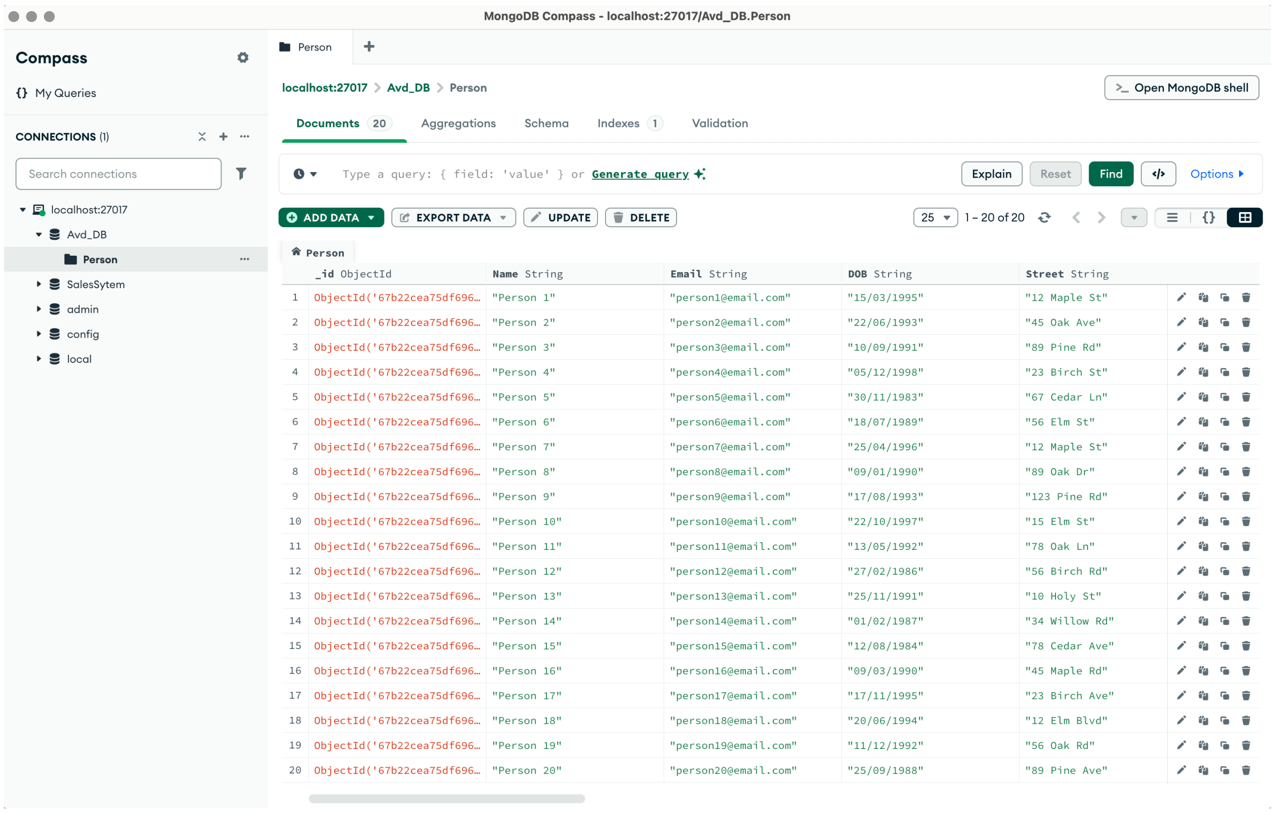
☑ Ignore empty strings

☐ Stop on errors

Specify Fields and Types Learn more about data types

	☑ Name	☑ Email	☑ DOB	☑ Street	☑ City	☑ C
	String	String	String	String	String	Str
1	Person 1	person1@email.com	15/03/1995	12 Maple St	London	Engl
2	Person 2	person2@email.com	22/06/1993	45 Oak Ave	Manchester	Engl
3	Person 3	person3@email.com	10/09/1991	89 Pine Rd	Birmingham	Engl
4	Person 4	person4@email.com	05/12/1998	23 Birch St	Edinburgh	Scot

THE PERSON COLLECTION



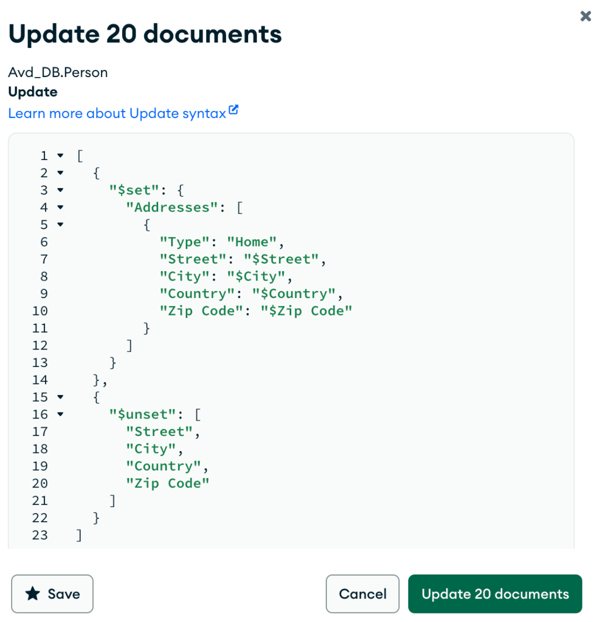
	_id ObjectId	Name String	Email String	DOB String	Street String
1	ObjectId('67b22cea75df696…	"Person 1"	"person1@email.com"	"15/03/1995"	"12 Maple St"
2	ObjectId('67b22cea75df696…	"Person 2"	"person2@email.com"	"22/06/1993"	"45 Oak Ave"
3	ObjectId('67b22cea75df696…	"Person 3"	"person3@email.com"	"10/09/1991"	"89 Pine Rd"
4	ObjectId('67b22cea75df696…	"Person 4"	"person4@email.com"	"05/12/1998"	"23 Birch St"
5	ObjectId('67b22cea75df696…	"Person 5"	"person5@email.com"	"30/11/1983"	"67 Cedar Ln"
6	ObjectId('67b22cea75df696…	"Person 6"	"person6@email.com"	"18/07/1989"	"56 Elm St"
7	ObjectId('67b22cea75df696…	"Person 7"	"person7@email.com"	"25/04/1996"	"12 Maple St"
8	ObjectId('67b22cea75df696…	"Person 8"	"person8@email.com"	"09/01/1990"	"89 Oak Dr"
9	ObjectId('67b22cea75df696…	"Person 9"	"person9@email.com"	"17/08/1993"	"123 Pine Rd"
10	ObjectId('67b22cea75df696…	"Person 10"	"person10@email.com"	"22/10/1997"	"15 Elm St"
11	ObjectId('67b22cea75df696…	"Person 11"	"person11@email.com"	"13/05/1992"	"78 Oak Ln"
12	ObjectId('67b22cea75df696…	"Person 12"	"person12@email.com"	"27/02/1986"	"56 Birch Rd"
13	ObjectId('67b22cea75df696…	"Person 13"	"person13@email.com"	"25/11/1991"	"10 Holy St"
14	ObjectId('67b22cea75df696…	"Person 14"	"person14@email.com"	"01/02/1987"	"34 Willow Rd"
15	ObjectId('67b22cea75df696…	"Person 15"	"person15@email.com"	"12/08/1984"	"78 Cedar Ave"
16	ObjectId('67b22cea75df696…	"Person 16"	"person16@email.com"	"09/03/1990"	"45 Maple Rd"
17	ObjectId('67b22cea75df696…	"Person 17"	"person17@email.com"	"17/11/1995"	"23 Birch Ave"
18	ObjectId('67b22cea75df696…	"Person 18"	"person18@email.com"	"20/06/1994"	"12 Elm Blvd"
19	ObjectId('67b22cea75df696…	"Person 19"	"person19@email.com"	"11/12/1992"	"56 Oak Rd"
20	ObjectId('67b22cea75df696…	"Person 20"	"person20@email.com"	"25/09/1988"	"89 Pine Ave"

UPDATING DOCUMENTS

The data is transformed into a more structured design to improve efficiency:

- The imported CSV file had a flat structure, which made address details, favourite items and neighbours' information scattered as separate fields. By grouping related data into embedded subdocuments, the data is more readable and organised.
- This design simplifies updates and maintenance, as the entire subdocument can be updated with a single query, reducing redundancy and complexity.
- With this design, queries become more efficient, as MongoDB can fetch entire objects instead of selecting multiple fields when retrieving data. This also improves indexing performance of traversing the nested documents rather than scanning unrelated fields.
- This architecture improves the database's flexibility and scalability for future expansion.

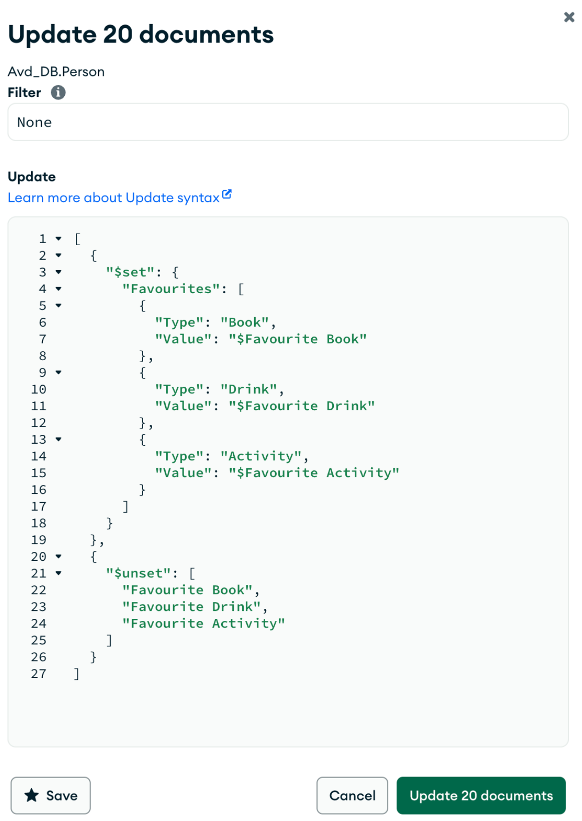
UPDATING ADDRESSES



```
[
  {
    "$set": {
      "Addresses": [
        {
          "Type": "Home",
          "Street": "$Street",
          "City": "$City",
          "Country": "$Country",
          "Zip Code": "$Zip Code"
        }
      ]
    }
  },
  {
    "$unset": [
      "Street",
      "City",
      "Country",
      "Zip Code"
    ]
  }
]
```

Instead of having different fields for *Street*, *City*, Country and *Zip Code*, the address details are reconstructed into an array of objects called Address as well as the *type* of address for accommodating new types of addresses such as home, work, etc and different addresses for each person.

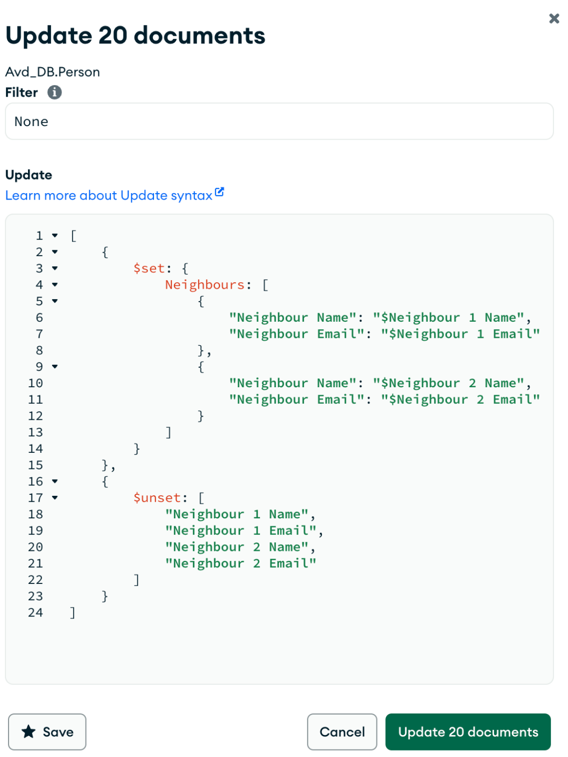
UPDATING FAVOURITES



```
[
  {
    "$set": {
      "Favourites": [
        {
          "Type": "Book",
          "Value": "$Favourite Book"
        },
        {
          "Type": "Drink",
          "Value": "$Favourite Drink"
        },
        {
          "Type": "Activity",
          "Value": "$Favourite Activity"
        }
      ]
    }
  },
  {
    "$unset": [
      "Favourite Book",
      "Favourite Drink",
      "Favourite Activity"
    ]
  }
]
```

To allow the system to add different types of favourites and many favourites for each kind, all favourite types are reconstructed into an array of objects named *Favourites* rather than having separate entries for *Favourite Book*, *Favourite Drink*, and *Favourite Activity*.

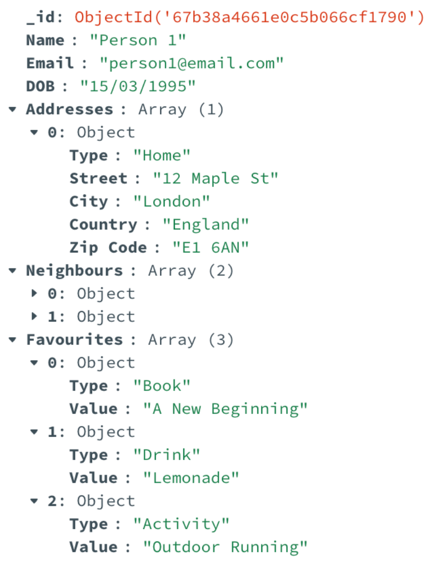
UPDATING NEIGHBOURS



```
[
    {
        $set: {
            Neighbours: [
                {
                    "Neighbour Name": "$Neighbour 1 Name",
                    "Neighbour Email": "$Neighbour 1 Email"
                },
                {
                    "Neighbour Name": "$Neighbour 2 Name",
                    "Neighbour Email": "$Neighbour 2 Email"
                }
            ]
        }
    },
    {
        $unset: [
            "Neighbour 1 Name",
            "Neighbour 1 Email",
            "Neighbour 2 Name",
            "Neighbour 2 Email"
        ]
    }
]
```

All neighbour information is reconstructed into an array of objects named *Neighbours* rather than having separate entries for each neighbour, allowing the system to add several neighbours for each individual without altering the entire collection.

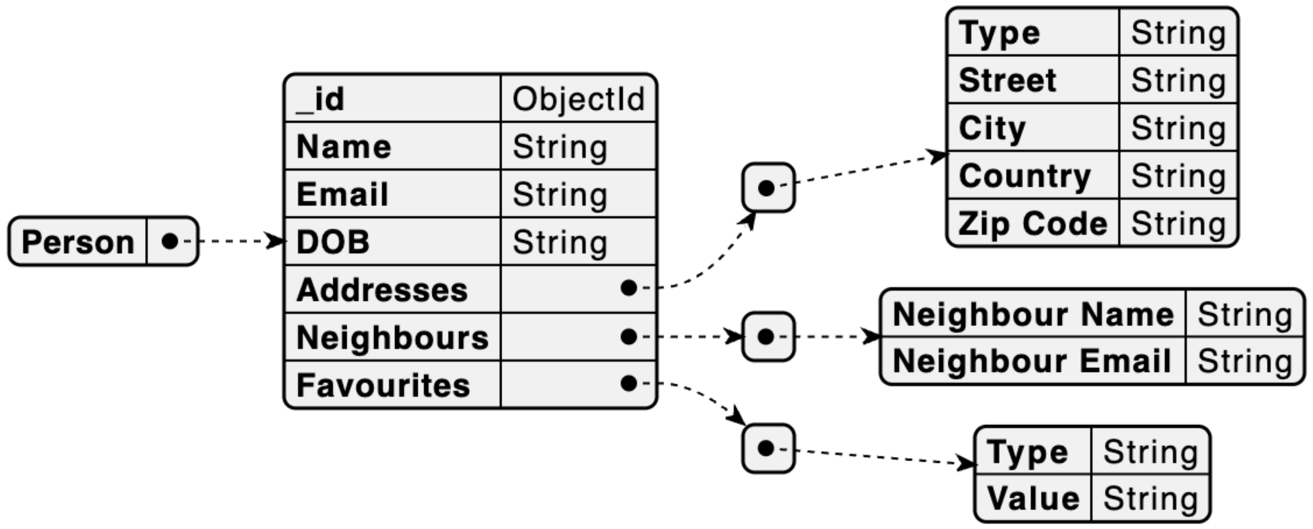
EXAMPLE OBJECT AFTER ALL UPDATES



```
_id: ObjectId('67b38a4661e0c5b066cf1790')
Name : "Person 1"
Email : "person1@email.com"
DOB : "15/03/1995"
Addresses : Array (1)
  0: Object
    Type : "Home"
    Street : "12 Maple St"
    City : "London"
    Country : "England"
    Zip Code : "E1 6AN"
Neighbours : Array (2)
  0: Object
  1: Object
Favourites : Array (3)
  0: Object
    Type : "Book"
    Value : "A New Beginning"
  1: Object
    Type : "Drink"
    Value : "Lemonade"
  2: Object
    Type : "Activity"
    Value : "Outdoor Running"
```

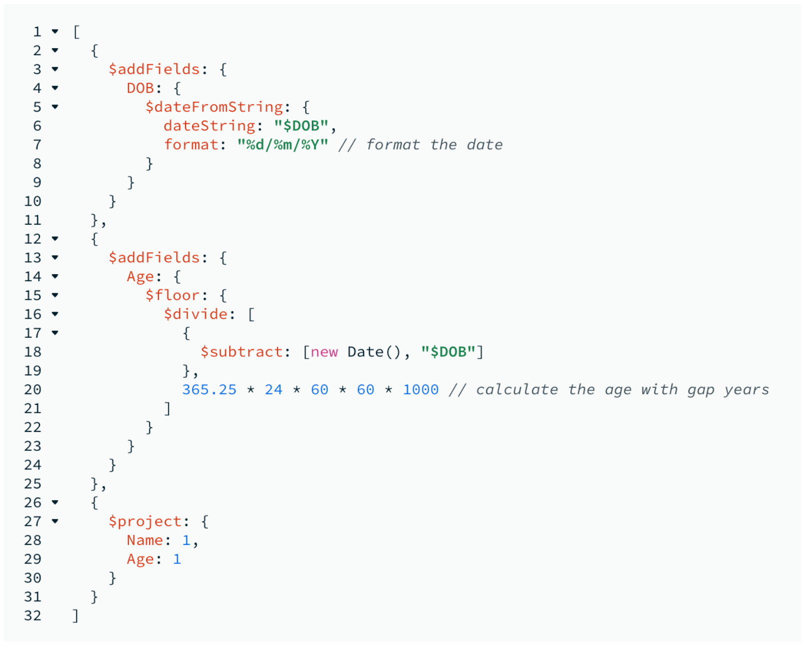
DOCUMENT DATA MODEL

The Document Data Model is created using PlantUML as below:

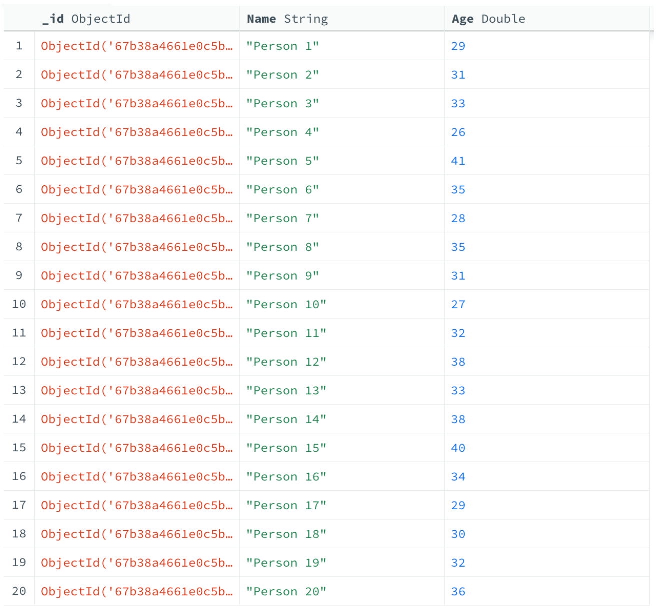


QUERIES

QUERY 1: DISPLAY ALL PERSONS' NAME AND THEIR AGES IN YEARS

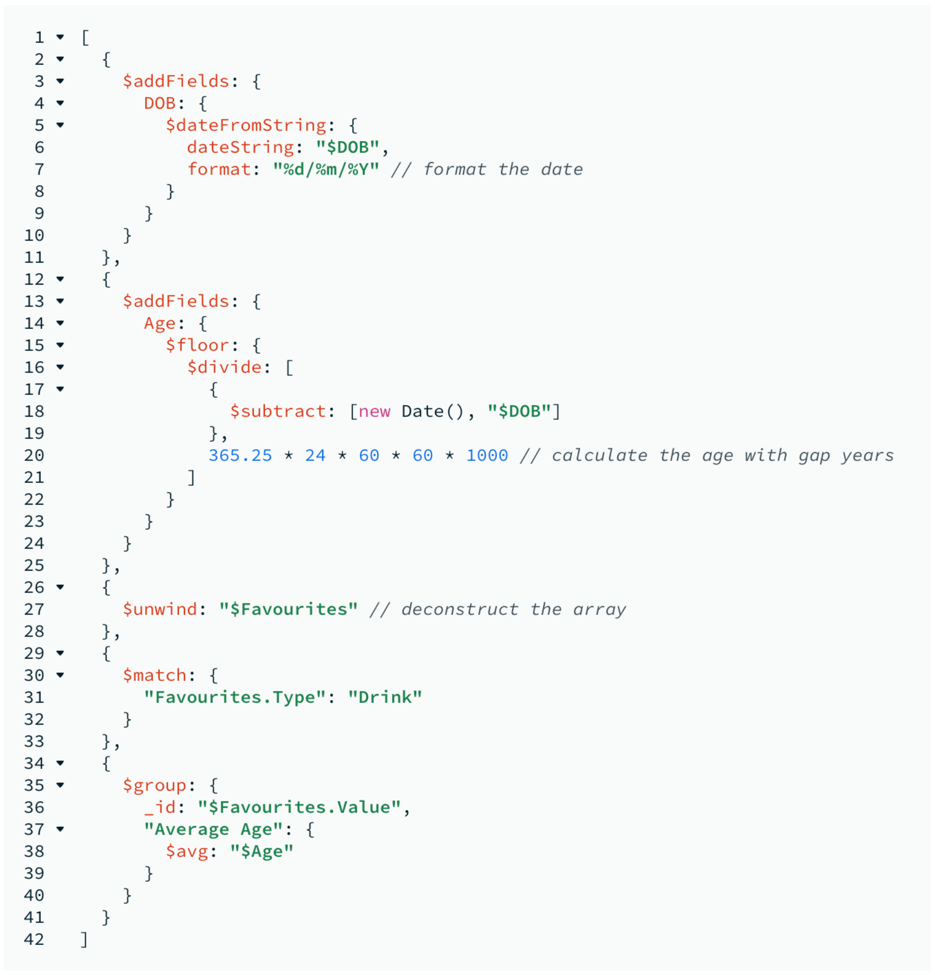


```
[
  {
    $addFields: {
      DOB: {
        $dateFromString: {
          dateString: "$DOB",
          format: "%d/%m/%Y" // format the date
        }
      }
    }
  },
  {
    $addFields: {
      Age: {
        $floor: {
          $divide: [
            {
              $subtract: [new Date(), "$DOB"]
            },
            365.25 * 24 * 60 * 60 * 1000 // calculate the age with gap years
          ]
        }
      }
    }
  },
  {
    $project: {
      Name: 1,
      Age: 1
    }
  }
]
```

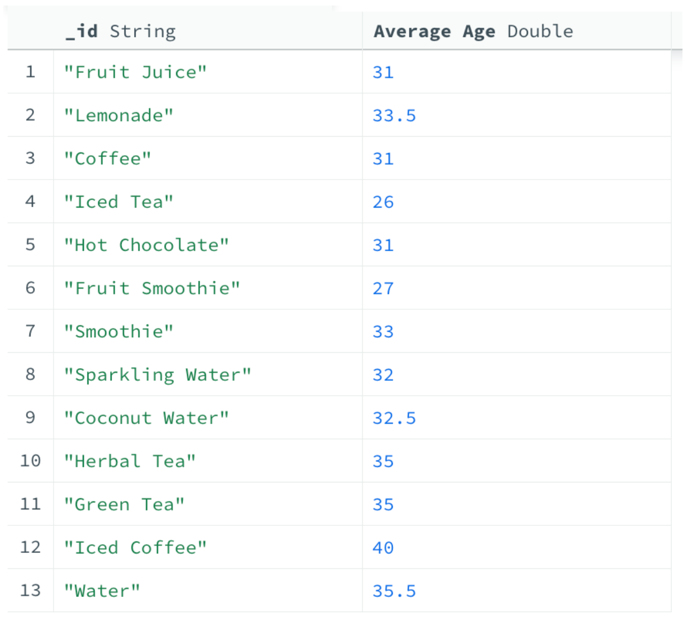


	_id ObjectId	Name String	Age Double
1	ObjectId('67b38a4661e0c5b…	"Person 1"	29
2	ObjectId('67b38a4661e0c5b…	"Person 2"	31
3	ObjectId('67b38a4661e0c5b…	"Person 3"	33
4	ObjectId('67b38a4661e0c5b…	"Person 4"	26
5	ObjectId('67b38a4661e0c5b…	"Person 5"	41
6	ObjectId('67b38a4661e0c5b…	"Person 6"	35
7	ObjectId('67b38a4661e0c5b…	"Person 7"	28
8	ObjectId('67b38a4661e0c5b…	"Person 8"	35
9	ObjectId('67b38a4661e0c5b…	"Person 9"	31
10	ObjectId('67b38a4661e0c5b…	"Person 10"	27
11	ObjectId('67b38a4661e0c5b…	"Person 11"	32
12	ObjectId('67b38a4661e0c5b…	"Person 12"	38
13	ObjectId('67b38a4661e0c5b…	"Person 13"	33
14	ObjectId('67b38a4661e0c5b…	"Person 14"	38
15	ObjectId('67b38a4661e0c5b…	"Person 15"	40
16	ObjectId('67b38a4661e0c5b…	"Person 16"	34
17	ObjectId('67b38a4661e0c5b…	"Person 17"	29
18	ObjectId('67b38a4661e0c5b…	"Person 18"	30
19	ObjectId('67b38a4661e0c5b…	"Person 19"	32
20	ObjectId('67b38a4661e0c5b…	"Person 20"	36

QUERY 2: GROUP PERSONS BY THEIR FAVOURITE DRINK AND RETURN AVERAGE AGE OF EACH GROUP

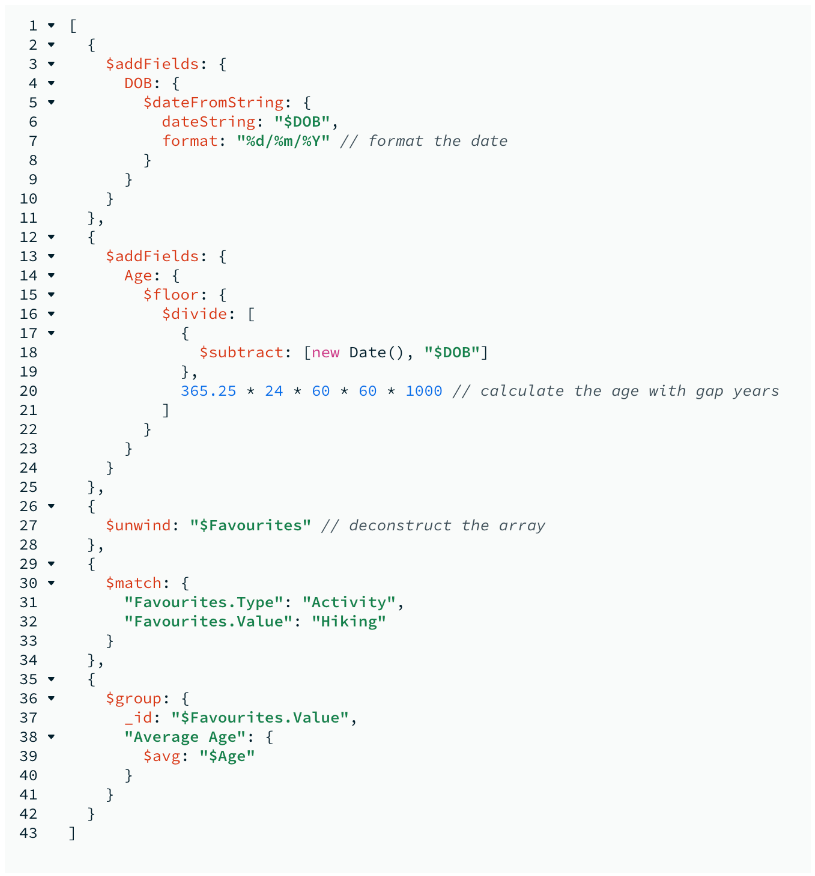


```
[
  {
    $addFields: {
      DOB: {
        $dateFromString: {
          dateString: "$DOB",
          format: "%d/%m/%Y" // format the date
        }
      }
    }
  },
  {
    $addFields: {
      Age: {
        $floor: {
          $divide: [
            {
              $subtract: [new Date(), "$DOB"]
            },
            365.25 * 24 * 60 * 60 * 1000 // calculate the age with gap years
          ]
        }
      }
    }
  },
  {
    $unwind: "$Favourites" // deconstruct the array
  },
  {
    $match: {
      "Favourites.Type": "Drink"
    }
  },
  {
    $group: {
      _id: "$Favourites.Value",
      "Average Age": {
        $avg: "$Age"
      }
    }
  }
]
```

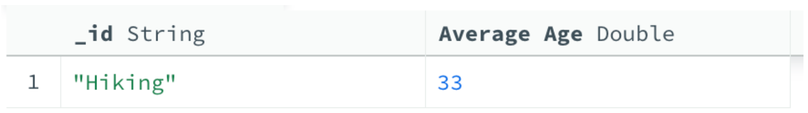


	_id String	Average Age Double
1	"Fruit Juice"	31
2	"Lemonade"	33.5
3	"Coffee"	31
4	"Iced Tea"	26
5	"Hot Chocolate"	31
6	"Fruit Smoothie"	27
7	"Smoothie"	33
8	"Sparkling Water"	32
9	"Coconut Water"	32.5
10	"Herbal Tea"	35
11	"Green Tea"	35
12	"Iced Coffee"	40
13	"Water"	35.5

QUERY 3: DISPLAY THE AVERAGE AGE OF PEOPLE WHO LIKE HIKING

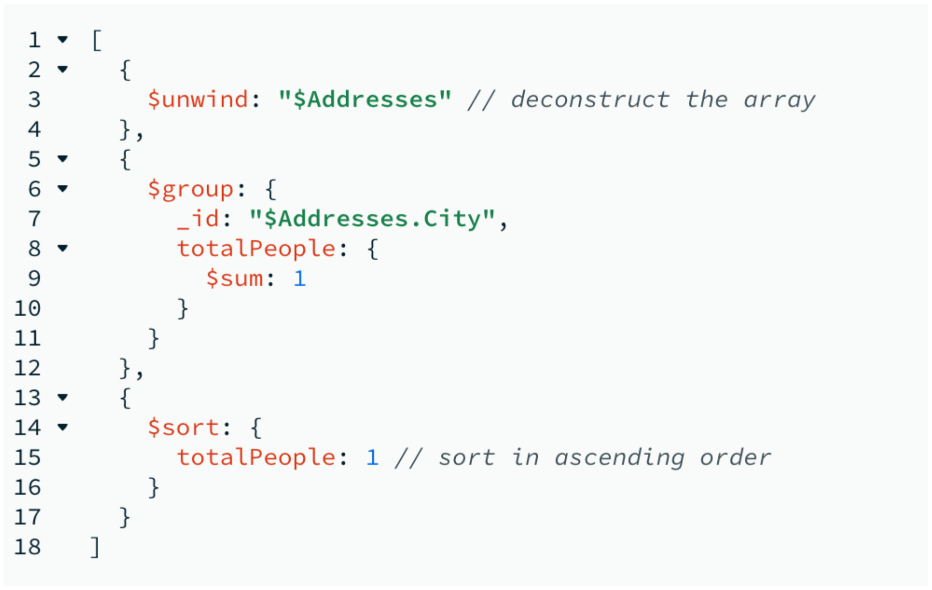


```
[
  {
    $addFields: {
      DOB: {
        $dateFromString: {
          dateString: "$DOB",
          format: "%d/%m/%Y" // format the date
        }
      }
    }
  },
  {
    $addFields: {
      Age: {
        $floor: {
          $divide: [
            {
              $subtract: [new Date(), "$DOB"]
            },
            365.25 * 24 * 60 * 60 * 1000 // calculate the age with gap years
          ]
        }
      }
    }
  },
  {
    $unwind: "$Favourites" // deconstruct the array
  },
  {
    $match: {
      "Favourites.Type": "Activity",
      "Favourites.Value": "Hiking"
    }
  },
  {
    $group: {
      _id: "$Favourites.Value",
      "Average Age": {
        $avg: "$Age"
      }
    }
  }
]
```

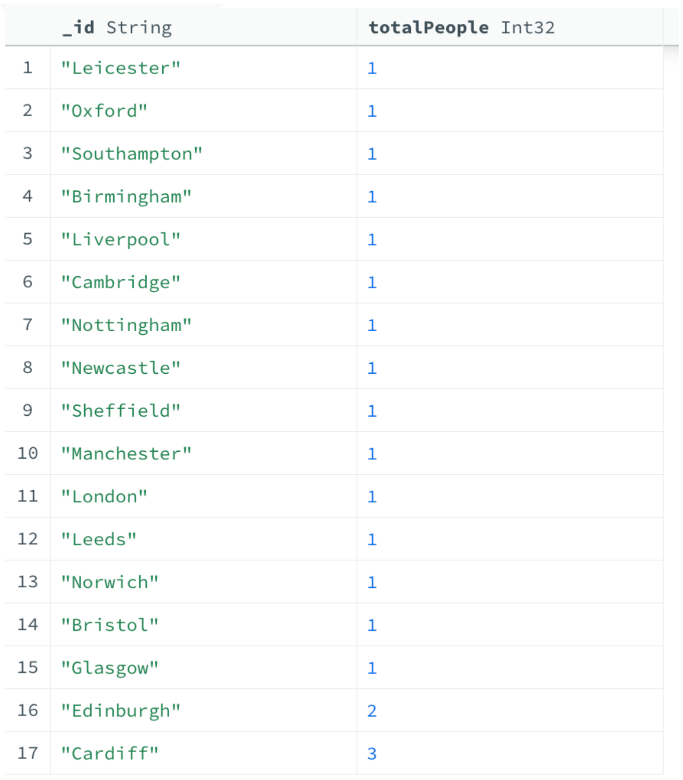


	_id String	Average Age Double
1	"Hiking"	33

QUERY 4: DISPLAY THE TOTAL NUMBER OF PEOPLE FROM EACH CITY AND SORT IT IN ASCENDING ORDER BY TOTAL NUMBER OF PEOPLE

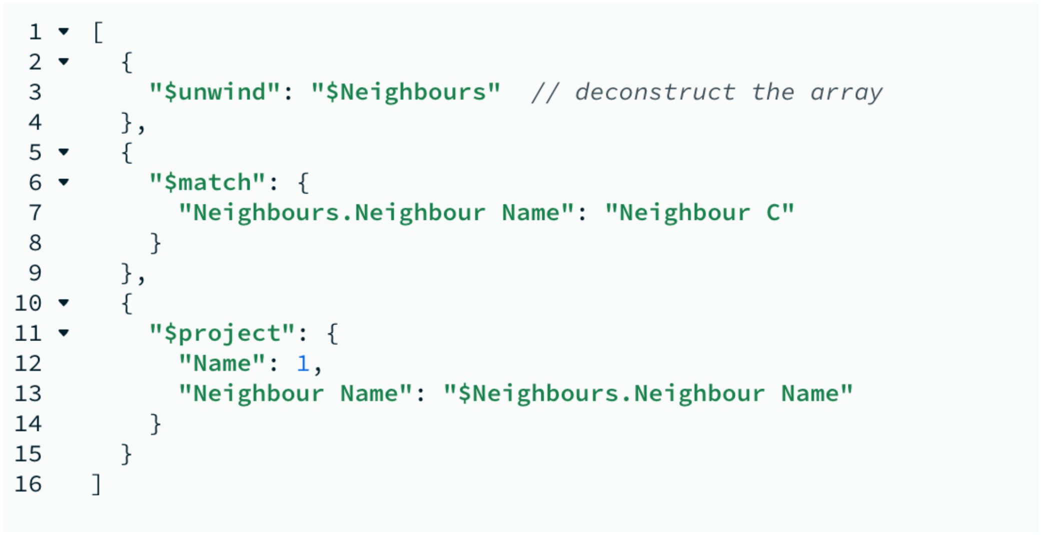


```
[
  {
    $unwind: "$Addresses" // deconstruct the array
  },
  {
    $group: {
      _id: "$Addresses.City",
      totalPeople: {
        $sum: 1
      }
    }
  },
  {
    $sort: {
      totalPeople: 1 // sort in ascending order
    }
  }
]
```

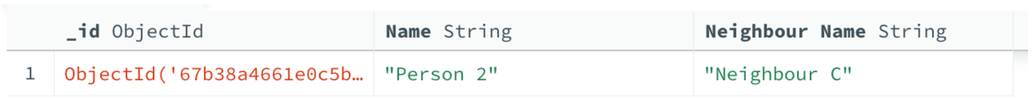


	_id String	totalPeople Int32
1	"Leicester"	1
2	"Oxford"	1
3	"Southampton"	1
4	"Birmingham"	1
5	"Liverpool"	1
6	"Cambridge"	1
7	"Nottingham"	1
8	"Newcastle"	1
9	"Sheffield"	1
10	"Manchester"	1
11	"London"	1
12	"Leeds"	1
13	"Norwich"	1
14	"Bristol"	1
15	"Glasgow"	1
16	"Edinburgh"	2
17	"Cardiff"	3

QUERY 5: DISPLAY NAME OF PERSON(S) WHOSE NEIGHBOUR IS NEIGHBOUR C



```
[
  {
    "$unwind": "$Neighbours"  // deconstruct the array
  },
  {
    "$match": {
      "Neighbours.Neighbour Name": "Neighbour C"
    }
  },
  {
    "$project": {
      "Name": 1,
      "Neighbour Name": "$Neighbours.Neighbour Name"
    }
  }
]
```



	_id ObjectId	Name String	Neighbour Name String
1	ObjectId('67b38a4661e0c5b…	"Person 2"	"Neighbour C"