Student Number: 2644995

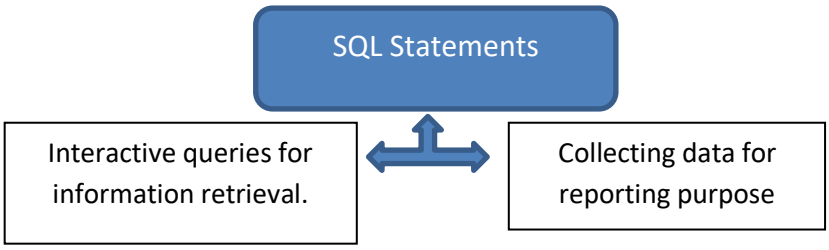
Table of Contents

Question 1: -

Part A: -

A Relational database means a database in which data is stored in a structured format using, rows and column which makes it easy to locate and access specific values within the database (without having to reorganize the tables). As the values in the table are related to each other, that is why, it is called as 'relational database'.

For database interaction, Structured Query Language that is a standard application, used by Relational Database to provide an Application User Interface (API).



This is a problem for a relational database because each item for sale has different features which will depend on the classification of that item. So suppose if we have 10 items classifications, than we will have to create 10 different tables to store their features, which will take many resources. In addition, we will have to add a new table, every time we want to add a new classification item.

1st Solution:

Create separate table for each classification of item. All the items will be stored in the same table. However, all the features of every item will be stored according to the classification of that item in a different table. Hence, it portrays that minimum space is required as there are only two tables and further, specification table contains null values. When storing data about items in specification table, the two main actions to be performed are selecting a number of columns and most number of columns have to be marked null.

2nd Solution:

We can create different tables for differently classified items. In this scenario, we will store all the clothes in a separate table with their respective features, all the cars in another table and so on.

Comparison of two Solutions

In the first solution, all the items will be stored in a single table. Therefore, we can simply refer to one table whenever we are searching for an item while in the second scenario we will have to keep track of the table in which items stored. We will have to look into only clothing table while searching for any clothing item. In the first solution, we will have to add a new table for each item classification. Therefore, every time a new classification of item is added, a new table will be created. In the second

scenario, a new classification will again result into a table, which will contain the entire item's data along with the classifications.

I would prefer the first solution, as all the item's common data is being stored in the same table and we will not need to keep a track of where to look for an item's data.

Part B: -

- **Primary key of each table and reasons behind our choice: -**

Table name	Primary key	Reasons behind the choice
USERS	USER_ID	This key uniquely identifies each user (Customer/ seller). I have used integer data type and auto increment. When we add new user in our database then it automatically generates the user id in integer.
ITEMS	ITEM_NO	This key uniquely identifies each item. For example, an item bag code is 101.
SALE	SALE_NO	This key uniquely identifies each sale that is Advertised and sold by seller.
TRANSACTION	TRA_NO	This key Uniquely identifies each transaction done by customer. When customer do transaction then system automatically generate a transaction number.it can be seller or buyer Customer
ORDER_BY	S_NO	This key uniquely identifies each item purchased by the Customer.
MESSAGES	MESSAGE_NO	This key Uniquely identifies each message sent to a customer corresponding to a customer. When transaction is done, one or many messages are recorded for a customer. In such cases then it automatically creates this key.

User table has information about the customer, which could be either buyer or seller. This table has a relationship with Sale and Transaction tables by one-to-many relationship, that means there could be many transactions by a user and user can purchase more than one items. Primary key for this table is U_Id, and it does not have any foreign key.

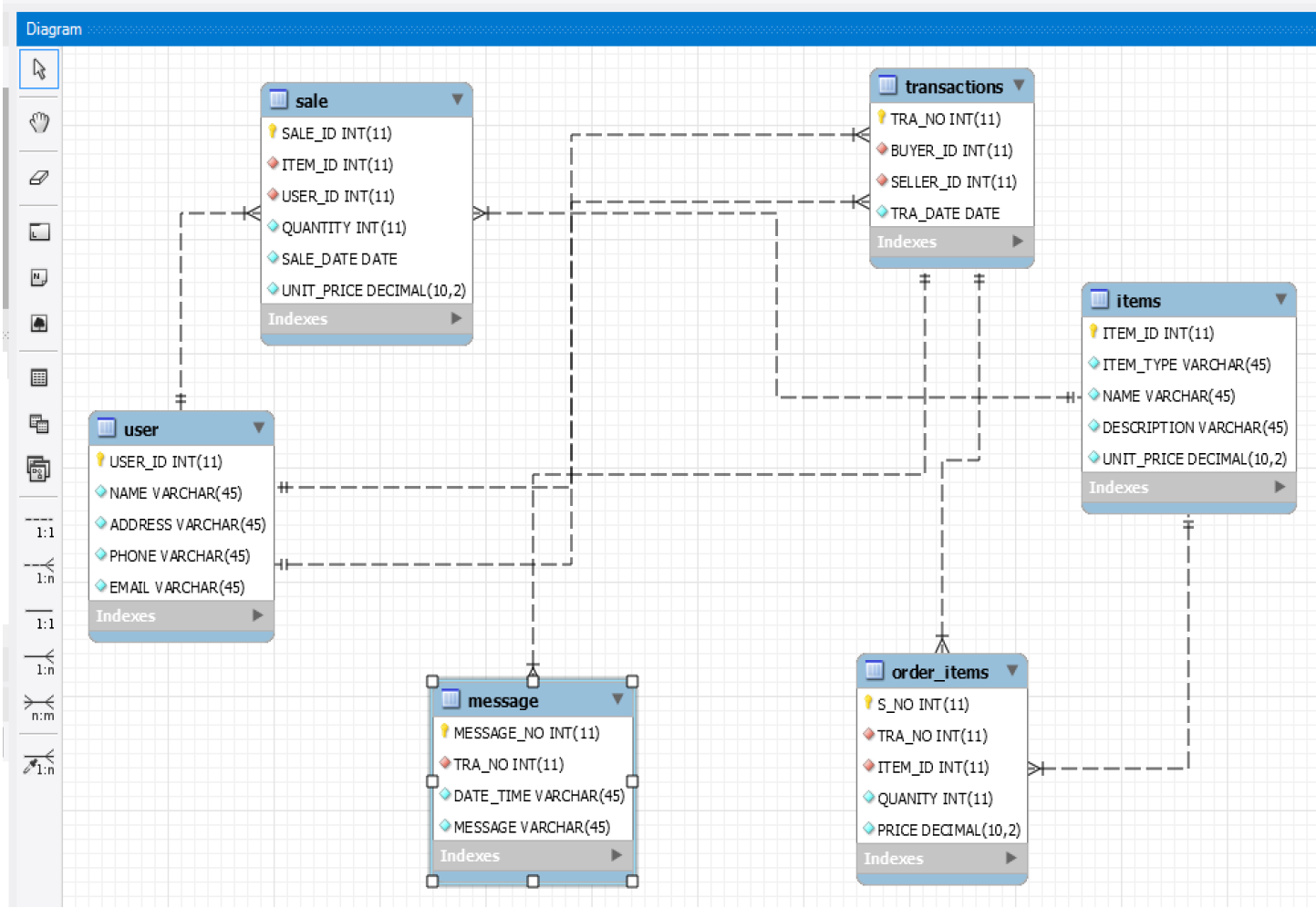
Sale table has information about initial selling process as well as final process e.g. advertisement of items and list of sold items by seller. It has one-to-many-relationship with Items table, which means that one item can have one advertisement. Sale_No is the primary key in this table, while Item_id and User_id are the foreign keys for this table.

Order_items table has many-to-one and zero-to-many relationship (one item may not have any orders) to Transaction table. It contains product information e.g. item name, item t description, item price etc. S_No is the primary key for this table while Tran_no and Item_id are foreign keys referencing to transaction and item tables respectively.

Student Number: 2644995

Transaction table provides information about type of user, which could be either buyer or seller. Relationship between transaction table and order table is one-to-many. For example, one user can order many items as a buyer. Tra_No is the primary key for this table while Buyer_id and Seller_id are foreign keys.

Message table stores the communication between buyer and seller that leads to transaction. Message_No is the unique identifier and the primary key for this table while Tran_No is referencing to transaction table and it is stored as foreign key.



Note :The pink dot means Foreign key ,and key shape is primary key

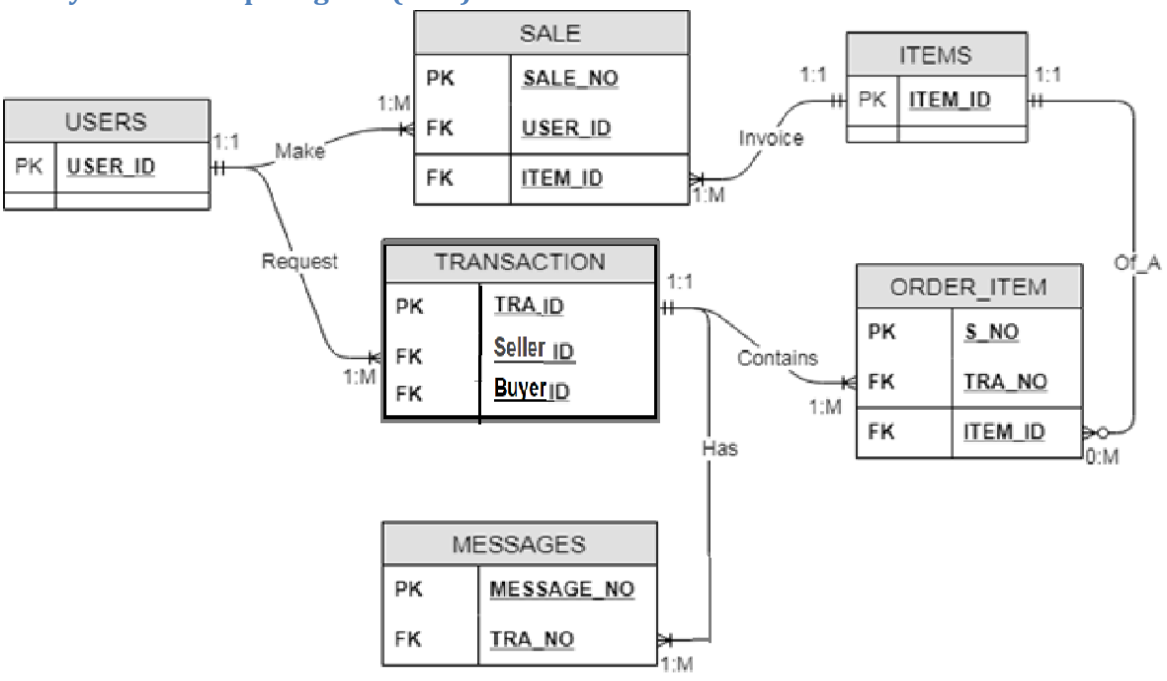
- **Normalization in 3NF: -**

I have normalized the database up to third normal form (3NF). Now this database does not contain the transitive dependency. For example USERS table. In this table user id is the primary key that provides the information of all other fields of User table. There is no other field that provides the information. So there is no transitive dependency and this table in third normal form.

I have normalized the database up to third normal (3NF) because in this form there is no duplicate data, no partial and transitive dependency and ensured about referential integrity.

Part c: -

Entity Relationship Diagram (ERD): -



Assumption: -

- A USER request to one or many TRANSACTIONS.
- Each TRANSACTION is related to one and only one USER.

- A USER sells/Buys many ITEMS.
  - A USER can make one or many SALES.
  - Each SALE is related to one and only one USER.
- An ITEM is being sold by many SELLERS.
    - An ITEM is related to one or many SALES.
    - Each SALE is related to one and only one ITEM.

- A TRANSACTION has one or many MESSAGES.
- Each MESSAGE is related to one and only one TRANSACTION.

- A TRANSACTION has many ITEMS.
  - A TRANSACTION contains one or many ORDER_ITEMS.
  - Each ORDER_ITEM is related to one and only one TRANSACTION.
- An ITEM has many TRANSACTIONS.
  - An ITEM has zero or many ORDER_ITEMS.
  - Each ORDER_ITEM is related to one and only one ITEM.

Tables and Data: -

```
/*----------------------------------------

----------CREATE TABLES----------

----------------------------------------*/

/*---1. USERS TABLE----------*/

CREATE TABLE Users

(

U_Id INT AUTO_INCREMENT PRIMARY KEY,

Name VARCHAR(30) NOT NULL,

Address VARCHAR(200) NOT NULL,

Phone VARCHAR(16) NOT NULL,

Email VARCHAR(45)
```

Student Number: 2644995

```
);



/*-----2.. ITEMS TABLE----------*/
CREATE TABLE ITEMS
(
I_Id INT PRIMARY KEY,
Item_Type VARCHAR(20) NOT NULL,
Name VARCHAR(30) NOT NULL,
Item_Describption VARCHAR(500),
Quantity INT NOT NULL,
Price DECIMAL(10,2)
);
/*------3. SALE TABLE----------*/
CREATE TABLE Sales
(
Sale_NO INT AUTO_INCREMENT PRIMARY KEY,
U_ID INT,
ITEM_ID INT,
Quantity INT NOT NULL,
Price DECIMAL(10,2),
Sale_Date DATE NOT NULL,
FOREIGN KEY (U_Id) REFERENCES USERS(U_Id),
FOREIGN KEY (I_Id) REFERENCES ITEMS (I_Id)
);
```

```
/*--------4. TRANSACTIONS TABLE-----------*/
CREATE TABLE TRANSACTIONS
(
T_No INT AUTO_INCREMENT PRIMARY KEY,
Seller_Id INT,
Buyer_Id INT,
T_Date DATE NOT NULL,
FOREIGN KEY (Seller_Id) REFERENCES USERS(U_Id),
FOREIGN KEY (Buyer_Id ) REFERENCES USERS(U_Id)
);

/*-------5. ORDER_ITEMS TABLE-----------*/
CREATE TABLE ORDER_ITEMS
(
S_No INT AUTO_INCREMENT PRIMARY KEY,
T_No INT,
I_Id INT,
Quantity INT NOT NULL,
Price DECIMAL(10,2) NOT NULL,
FOREIGN KEY (T_No) REFERENCES TRANSACTIONS(T_No),
FOREIGN KEY (I_Id) REFERENCES ITEMS (I_Id)
);
```

```
CREATE TABLE MESSAGES
(
M_No INT AUTO_INCREMENT PRIMARY KEY,
T_No INT,
Date_Time DATETIME NOT NULL,
Message VARCHAR(255) NOT NULL,
FOREIGN KEY (T_No) REFERENCES TRANSACTIONS(T_No)
);
/*-------------------------------------------------------
-------------------------INERT DATA ----------------------
------------------------------------------------------------*/
/*---1. USERS TABLE-----------*/
INSERT INTO USERS(Name, Address, Phone, Email) VALUES
('Robin','US','6787678761','Robin123@gmail.com'),
('Kabin','US','6787678762','Kabin123@gmail.com'),
('Raan','US','6787678763','Raan123@gmail.com'),
('Caan','US','6787678764','Caan123@gmail.com'),
('Yoyo','US','6787678765','Yoyo123@gmail.com');
```

Student Number: 2644995

```
/*-----2. ITEMS TABLE----------*/
INSERT INTO ITEMS VALUES
(1,'ABC','PEN','',30,9.10),
(2,'YZ','COPY','',30,50.10),
(3,'BNJ','PENT','',30,500.10),
(4,'IJO','CHAIR','',30,1000.10),
(5,'NJM','TOP','',30,1100.10);

/*------3. SALE TABLE----------*/
INSERT INTO SALE(U_Id, I_Id, Quantity, Price, SALE_DATE) VALUES
(3,1,4,5.50,'2010-01-01'),
(4,2,4,10.50,'2010-01-01'),
(3,3,4,400.50,'2010-01-01'),
(4,4,4,500.50,'2010-01-01'),
(5,5,4,600.50,'2010-01-01');

/*--------4. TRANSACTIONS TABLE----------*/
INSERT INTO TRANSACTIONS (U_Id, TRA_DATE) VALUES
(1, '2010-12-01'),
(2, '2010-12-02'),
(1, '2010-12-03'),
(2, '2010-12-04'),
(1, '2010-12-05');

/*-------5. ORDER_ITEMS TABLE----------*/
```

```
INSERT INTO ORDER_ITEMS (T_No, I_Id, Quantity, Price) VALUES
(1,1,10,20.00),
(2,2,10,60.00),
(3,3,10,1000.00),
(4,4,10,1500.00),
(5,5,10,1500.00);

/*---------6. MESSAGES TABLE----------*/
INSERT INTO MESSAGES (T_No, DATE_TIME, Message) VALUES
(1, '2010-12-01 23:59:59','GHJK'),
(2, '2010-12-02 23:59:59','DFGH'),
(3, '2010-12-03 23:59:59','FGHJK'),
(4, '2010-12-04 23:59:59','CVBN'),
(5, '2010-12-05 23:59:59','XCVBN');
```

Part D: -

If the item ID is known, Return the Seller and Buyer Name with known item Id.

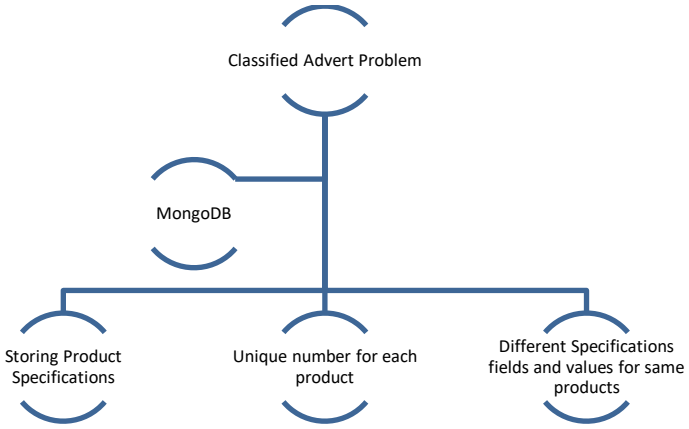
```
SELECT u1.NAME AS "CUSTOMER NAME", u2.NAME AS "SELLER NAME"
FROM USERS u1, USERS u2, ITEMS, SALE, TRANSACTIONS, ORDER_ITEMS
WHERE u1.USER_ID=TRANSACTIONS.U_Id
AND TRANSACTIONS.T_No=ORDER_ITEMS.T_No
AND ITEMS.I_Id=ORDER_ITEMS.I_Id
AND ITEMS.I_Id=SALE.I_Id
AND u2.U_Id=SALE.U_Id
AND ITEMS.I_Id=1;
```

Question 2: -

Part a: -

For classified advert problem, MongoDB offers a best solution to the because of its ability to derive a document-based data model. In a document database, all the data is stored in XML/JSON formats. We can store data with different attributes in a document database. There may be collection of documents having different features in a database whereas some other documents could have different features, MongoDB holds such kind of data in a very rich way.

For classified advert problem, it is suitable because it helps to solve a major problem of storing product specifications as more than one value can be stored in a single field. It solves the problem of having common fields or values for each product (i.e. unique number for each product). Also, different specifications fields and values can be defined for same products which can be further related with each other using 'Categories' field (for instance).



In our problem, as each item have different features. So, the features key in the document will contain different labels in different items. For the clothing items, the features will contains Size etc. whereas for cars, the same feature will contain model, make etc.

Part b: -

Pursuant to the classified advert database, the two possible use cases that can be performed are stated as follows:

Use Case 1 [Person with largest number of objects (Cars)]

We may want to find the person who has bought maximum number of objects. In this case, we will have to find the total count of items that each person has purchased and then find the person who has the highest count. So, the data will be fetched from People and Object tables and this task will be not carried out more often.

Use Case 2[List all Objects (Cars) having a particular classification]

The user may want to find all the objects that have a particular classification. In such a case, the data will be fetched from classification and object. This task can be carried out more often.

Part c: -

Aggregate (derived from Domain-Driven Design) is an abstraction that consists of related objects. These objects can be studied as a unit. Aggregation operations return a single result by performing variety of operations on the grouped data that is, compiled from multiple documents together.

In SQL, count(*) and with group by are used (similar to MongoDB aggregation) that gives significant meaning to the single value returned by aggregation operations.

People:

The aggregate that contains the information about each user of a website and consists of the fields like name, phone number, e-mail address and user identification et al. is known as People. It is a collection of documents which contains such information.

Based on the information fetched about the user, documents can be divided among clusters to use People as aggregate making people from same areas easy to reach as well as the system more efficient.

Object:

Like People, an Object is also a collection of documents which consists of information like price of the product, name of the product, warranty of the product, et al. pursuant to the information stored about products.

Due to lack of the information like specific address where the product is stored in a database, using Object will be expensive and time consuming because of searching all the products through each cluster.

Classification:

It refers to the grouping of products, which have standard features into single category. It can be achieved by storing all the data belonging to same category into one cluster, which will further make the system more user-friendly from user experience point of view by producing relevant results in terms of product search performed by the users.

Referring to Case 1 and Case 2, This means that we can either keep all the objects together(Cluster) for a person or we can keep all the objects according to their classification. All this will be decided according to what and how do we need to fetch the data.

If we will store all the products together for a person, then it will be very easy for us to find the person who has bought maximum number of objects but for getting all the objects under a particular classification, we will have to traverse all the person objects and operate on the data.

If we will store all the objects under a classification, then we will have some issues to find the largest number of objects for a person, but it will be very easy to fetch the objects under a particular classification.

Part d: -

As MongoDB does not have fixed data structure, therefore, it is a schema less database, which means that the data that gets stored in document database holds different documents that will further have different number of fields and values. Due to MongoDB's nature of incorporating varying sets of fields with different types for each field, one can use different number of fields and values making it schema less and more transparent and automatic time adjustments in the database.

It also means that it cannot support primary or foreign key like relational databases. We can use document validator to impose certain restrictions on the fields of our document. We can add certain restrictions like certain fields will have in it, string or double value.

These restrictions can be imposed with the help of document validator.

We have certain advantages like ensuring the data correctness with the help of validator i.e. we will have consistent data. However, we will have to update it as and when required according to our requirements.

Example:

```
db.createCollection("item"), {
  validator: {
    $jsonSchema: {
     bsonType: "object",
       required: ["serial number", "manufacturer ", "name", "price"],
        properties: {
          serial number: {
            bsonType: "int",
             minimum: 1,
             maximum: 20,
            description: "serial number must be a integer and is required"
            },
          manufacturer: {
             bsonType: "string",
             description: "manufacturer must be a string and is required"
             },
          name: {
             bsonType: "string",
             description: "product/item name must be a string and is required"
```

```
            },
          price: {
            bsonType: “int”,
            minimum: 10,
            maximum: 5000,
            },
        },
      },
    },
```

Part e: -

Indexes in MongoDB, are responsible for supporting the efficient execution of queries as they are specialized data structures and help in storing the value of specific field or set of fields ordered by the value of the field. We can create index on item name as we mostly use item's name while making a search for any item. Indexing helps a lot in making the search queries fasters. If we have created indexes on some fields, the performance can be boosted by the database engine by using those indexes in a number of different queries which will in turn helps in making the speed of our site faster.

We can't simply create indexes on each field as indexes take up space in memory. Too many indexes mean that the DB is going to swap them to and from disk. They also will increase insert and update time as indexes will be automatically updated by the DB in such cases.

Part f: -

Sharding means that the database will be stored on multiple servers instead of only a single server supporting high throughput operations as CPU capacity of the server is exhausted by the high query rates. Using sharding, each server will have a small bit of very large database on it. We can use item type as the shard key which means that which servers to store the data will be decided on the type of item. Each server will have data of certain items only and one item type cannot be stored on multiple servers.

I don't need any extra field to add as a shard key as we are using item type as the shard key and it is already present in my database design.

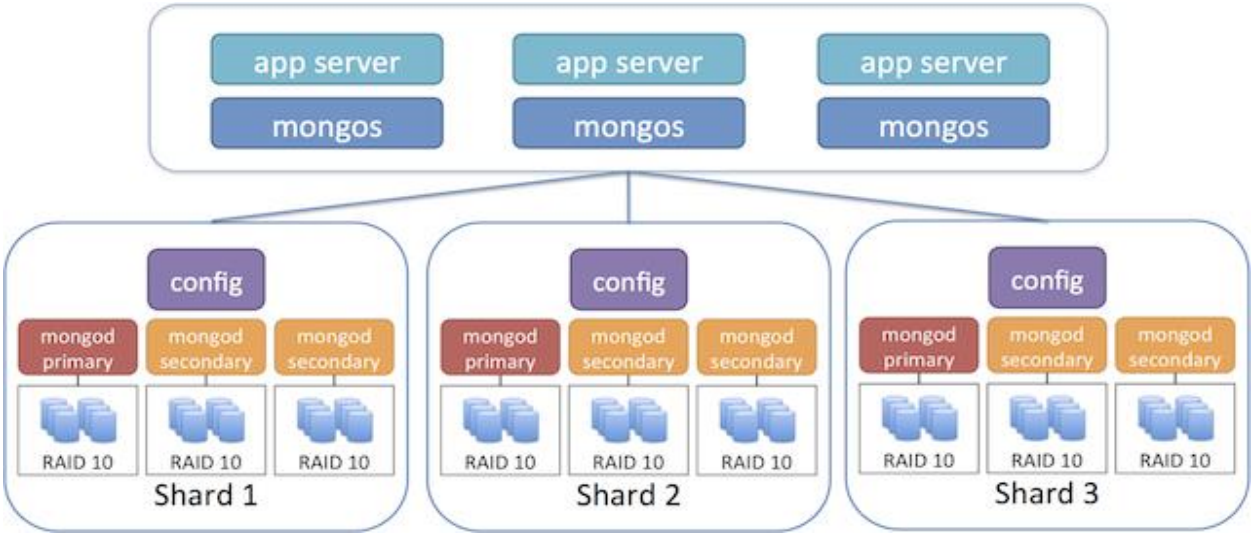


Figure: Sharding process in MongoDB
Source: https://mongodb-documentation.readthedocs.io/en/latest/ecosystem/tutorial/install-mongodb-on-amazon-ec2.html

Part g: -

First Option

1. We can have a messages array in each document, which will have the following labels:

Transaction No, Sender, Receiver, Message

Second Option:

2. We can have the messages array under each Person document, which will have the following labels:

Transaction Number, Receiver, Message

Third Option:

3. We can have messages array sender transaction document, which will contain the sender, receiver and the message data related to that particular transaction.

We will use the Second option, if we want to retrieve all the messages that a person has sent as, we can easily and directly fetch a person object and all the messages that he has sent will be fetched with it.

If we will use the first option, then we will have to traverse the whole array and fetch that message which has a particular person as the sender of message.

If we will use the third option, then we will have to go through every transaction to see that whether that person is present as a sender for that transaction or not.

Question 3: -

We will go with the NoSQL design for our classified advert website instead of SQL because of its certain advantages over the other approach.

1. MongoDB is horizontally scalable where as SQL is vertically scalable.i.e we can add more servers (sharing) in MongoDB but in RDBMS we can only increase the RAM.

2. Mongo DB supports a rich and expressive object model. These objects can have models and objects can be nested inside other objects. We can also index the property of an object at any level of the hierarchy.

3. We can use secondary indexes in NoSQL. Secondary indexes are a first-class construct in MongoDB. So we can easily index any property of an object even if it is a nested object.

4. NoSQL is a schema less model. In these databases, one collection holds different documents. But in RDBMS we have to first design the tables.

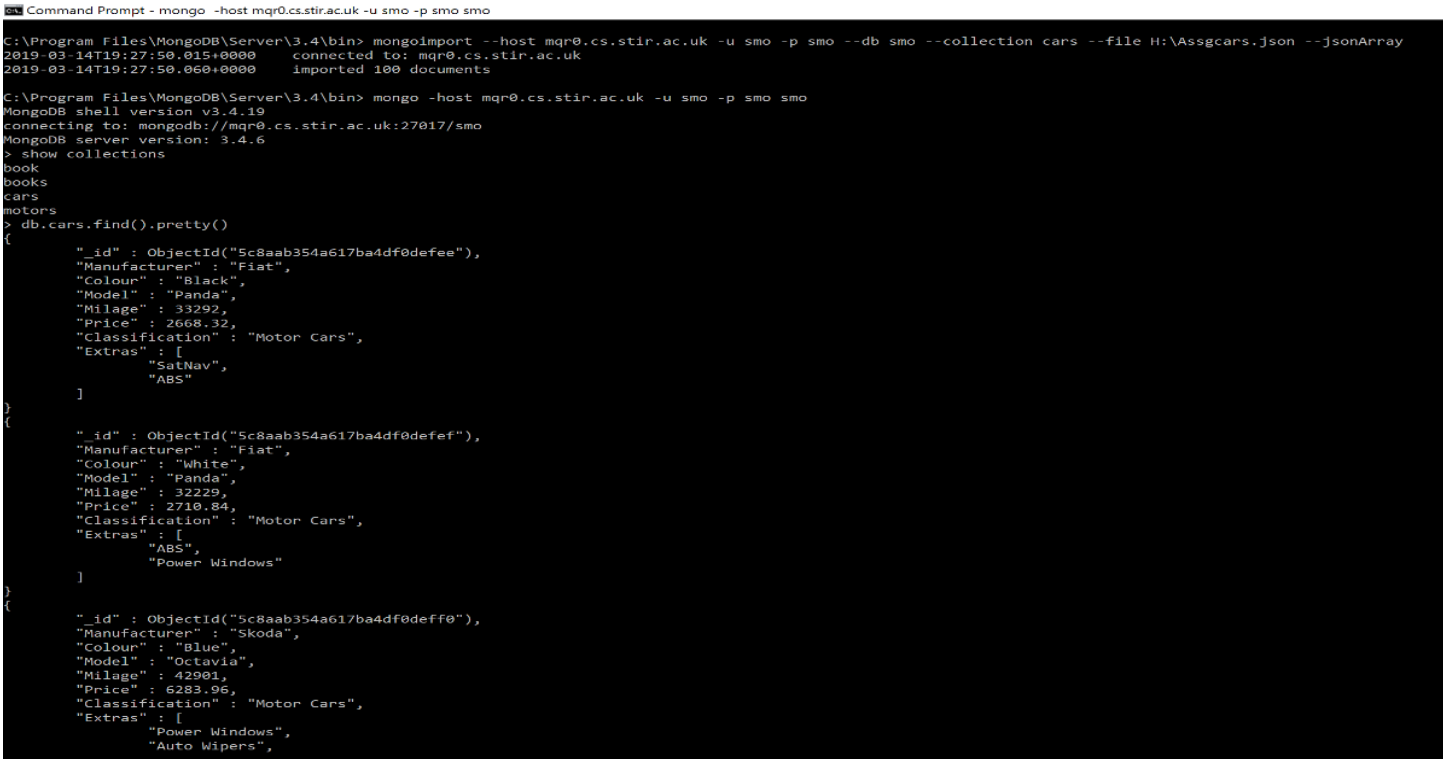
5. NoSQL is almost 100 times faster than RDBMS.

6. In NoSQL, Conversion and mapping from application objects to database objects is not required.

Question 4: -

Import json file into mongo

mongoimport --db smo --collection cars --jsonArray --file Assgcars.json



Part a:

db.cars.find ({"Manufacturer":"Skoda","Mo del":"Octavia"},{Price:1,_id:0});

```
> db.cars.find ({"Manufacturer":"Skoda","Model":"Octavia"},{Price:1,_id:0});
{ "Price" : 6283.96 }
{ "Price" : 5782.88 }
{ "Price" : 5906.24 }
{ "Price" : 5609.12 }
{ "Price" : 6025.68 }
{ "Price" : 5097.92 }
{ "Price" : 5754.68 }
{ "Price" : 5645 }
{ "Price" : 6283.52 }
```

Part b:

db.cars.aggregate([{$match: {"Manufacturer":"Skoda"}},{ $group: { _id:"Manufacturer", avgPrice:{ $avg:"$Price"}} }]);

```
>
>
> db.cars.aggregate([     {         $match: {"Manufacturer":"Skoda"}},     {         $group:         {             _id:"Manufacturer",             avgPrice:{ $avg:"$Price"}         }
{ "_id" : "Manufacturer", "avgPrice" : 5853.970370370371 }
>
```

Part c:

db.cars.aggregate({$match: {"Manufacturer":"Skoda"}}, { $group: { _id:"$Model", avgPrice:{ $avg:"$Price"} } }]);

```
> db.cars.aggregate([     {         $match: {"Manufacturer":"Skoda"}},     {         $group:         {             _id:"$Model",             avgPrice:{ $avg:"$Price"}         }
{ "_id" : "Yeti", "avgPrice" : 5945.356 }
{ "_id" : "Superb", "avgPrice" : 5940.006666666667 }
{ "_id" : "Fabia", "avgPrice" : 5287.3 }
{ "_id" : "Octavia", "avgPrice" : 5821 }
```

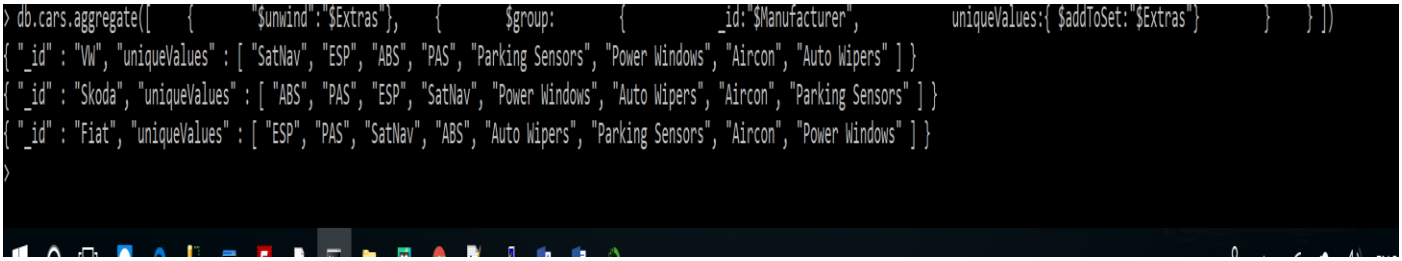
Part d:

db.cars.aggregate([{$match: {"Manufacturer":"Skoda"}},{$group: { _id:"$Model", avgPrice:{ $avg:"$Price"} } }, {$sort:{avgPrice:-1} },{$limit:1 }]);

```
> db.cars.aggregate([     {         $match: {"Manufacturer":"Skoda"}},     {         $group:         {             _id:"$Model",             avgPrice:{ $avg:"$Price"}         }     },     {         $sort:{avgPri
ce:-1}     },     {         $limit:1     } ]);
{ "_id" : "Yeti", "avgPrice" : 5945.356 }
>
```

Part e:

db.cars.aggregate([{"$unwind":"$Extras"}, {$group: {_id:"$Manufacturer",uniqueValues:{ $addToSet:"$Extras"} } }])


> db.cars.aggregate([{ "$unwind":"$Extras"}, { $group: { _id:"$Manufacturer", uniqueValues:{ $addToSet:"$Extras"} } }])
{ "_id" : "VW", "uniqueValues" : ["SatNav", "ESP", "ABS", "PAS", "Parking Sensors", "Power Windows", "Aircon", "Auto Wipers"] }
{ "_id" : "Skoda", "uniqueValues" : ["ABS", "PAS", "ESP", "SatNav", "Power Windows", "Auto Wipers", "Aircon", "Parking Sensors"] }
{ "_id" : "Fiat", "uniqueValues" : ["ESP", "PAS", "SatNav", "ABS", "Auto Wipers", "Parking Sensors", "Aircon", "Power Windows"] }
>