ITNPBD3 Assignment 2016

Name: Shazibali Gesawat

Student Number: 2530137

Design of a Database for a Classified Adverts Website

1.(a)

In the case of a Classified advert system database every category of advertisement has at least one feature that is not same as a different category. The problems caused in such a scenario is data redundancy and hence due to this duplication caused the cost of running that query is more than that of the optimized resources. Entering the same details for each row which increases the possibility of errors in entering data and will cause errors in retrieval of data. A poor design of a database is responsible for such errors.

The Possible Solutions for this problem are:

- Using a Relational Database such as MySQL for the Classification of data i.e. splitting of data into several series of tables which is known by the term normalization. Normalization of a database is most efficient way to design a relational database. Normalization is the process of identifying entities and their attributes, and defining the relationship between the entities.

Advantages:

- There is an ease in using the relational databases as everything is much clear and less complicated as compared to non-relational databases they are much more complicated document database.
- RDBMS truly provides all the ACID (Atomicity, Consistency, Isolation, Durability) properties as comparison with NoSQL does not truly offer these properties.
- Normalization and de-normalization is Efficient and less Complicated as compared to a non-relational database such as MongoDB.

Disadvantages:

- A major constraint and therefore disadvantage in the use of relational database system is machine performance. If the number of tables between which relationships to be established are large and the tables themselves effect the performance in responding to the SQL queries.
- It is very costly to set up as compared to non-relational databases such as mongo DB.

- The second solution in which we can solve such a scenario where we have multidimensional data is that we can use a non-relational, multidimensional document and a schema less database such as MongoDB which is a convenient approach for such a flexible database such as online advert system.

Advantages:

- MongoDB is a very fast Database.
- MongoDB is very flexible and adding up machines on MongoDB is very easy as the scale reads by using replica sets and writes by using sharding (auto balancing).
- Replication of data is easy through MongoDB.

Disadvantages:

- Data size in MongoDB is typically higher due to e.g. each document has field names stored it
- less flexibility with querying (e.g. no JOINs)
- No support for transactions - certain atomic operations are supported, at a single document level.
- less up to date information available because it is a fast evolving product

1(b). Database Design and in a Normalized Form i.e.2NF. (Second Normal Form).

User Table

User_ID(P.K)	User_Name	Password	Email	Phone	Address	City	Post Code
1010101	shazibali	******	ali@icloud.com	989898989	B street	Stirling	Fk8 1Na

PRIMARY KEY (**User_ID**): This Primary key makes it easier to access user information across different tables and with the help of JOIN it makes it easier to access desired information.

Buyer Table

User_ID(F.K)	Buyer_id (P.K)	Buyer_Name
1010101	123456	Shazib Ali

PRIMARY KEY (**Buyer_id**): The reason behind setting this as my primary key is making it available to the transactions table, user table, and Messages table. For example: In the Messages table messages are being transferred between buyer and seller regarding a product hence the seller is known to the buyer and vice versa.

Seller Table

Seller_id (P.K)	User_ID(F.K.)	Seller_Name
7616222	1010101	Express Guys

PRIMARY KEY (**Seller_id**):4 tables in this relational database need the instance of the Seller_id. The Seller needs to be known in instances such as Transactions, Messages and Product.

Product Table

Category_id(F.K)	Feature_id(F.K)	Product_id	Seller_id (F.K)	Name	Type	Price (in $)
124567	129299001	1890002	7616222	IPhone	New	2500

PRIMARY KEY (**Product_id**): Transactions need to know the product details and when Messaging is done between Buyer and Seller then the product details need to be known hence setting up Product_id as a primary key is necessary.

Category Table

Category_id(P.K)	Category_Name
120299	Motor Cars

PRIMARY KEY (**Category_id**): It helps in classifying the advert system efficiently based on categories and then it has its own features in the feature table.

Feature Table

Category_id(F.K)	Feature_id(P.K)	Feature_Name
120299	120299001	4x4 SUV

PRIMARY KEY(**Feature_id**): Every Category has different features and hence creating a feature table makes it easier to access data and reduces data redundancy.

Category_id: Foreign Key with References to *Category Table ->Category_id.*

Transaction Table

T_id (P.K)	Date/Time	Product_id	Seller_id	Buyer_id
17628800	2016-01-19 03:14:07	1890002	7616222	123456

PRIMARY KEY(**T_id**): Post Transaction if in a case a buyer needs to query the seller on the purchase of a product hence through the T_id the seller gets to know the details of the buyer and the product.

Product_id: Foreign Key with References to *Buyer Table->Buyer_id*

Seller_id: Foreign Key with references to *Seller Table ->Seller_id.*

Buyer_id: Foreign Key with References to *Buyer table->Buyer_id.*

Messages Table

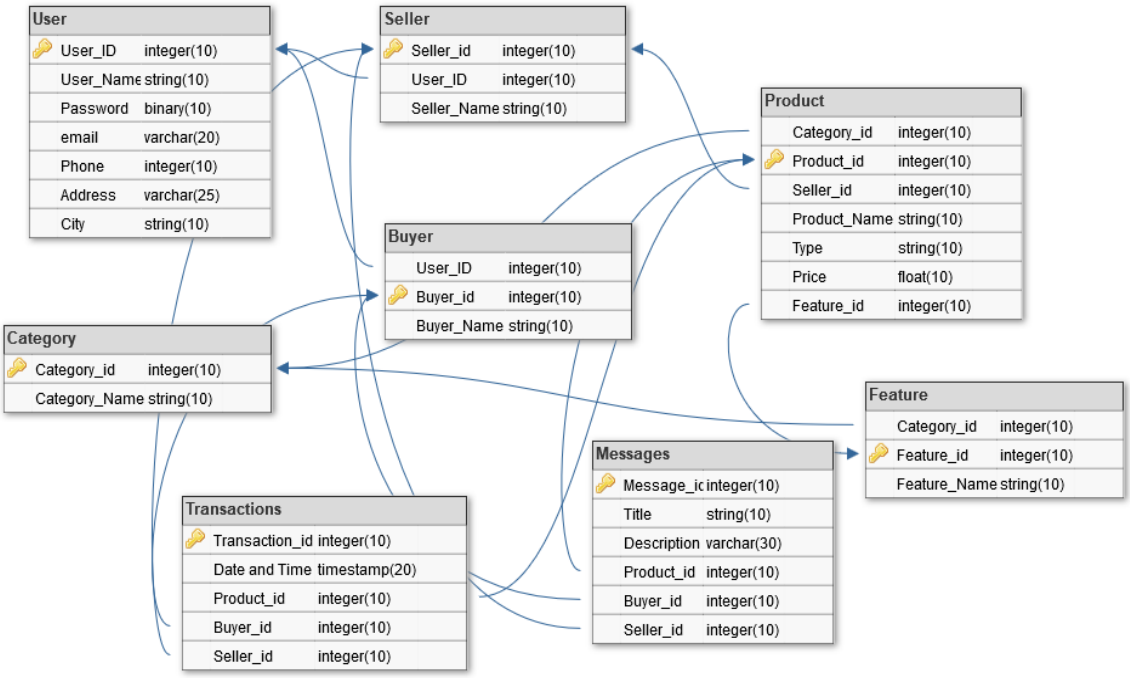
Message_id	Title	Description	Product_id	Seller_id	Buyer_id
0001922	IPhone	Regarding IPhone	1890002	7616222	123456

This database design is in the Second Normal Form and It is because there are no partial dependencies and the non-key attributes are fully dependent on the Primary Keys.

The Basic Reason behind Normalization is to avoid data redundancy. When there is no data redundancy hence less storage space is needed in the database after normalization and the data remains consistent. Normalization also avoids update/delete anomalies.

1.c)

ENTITY RELATIONSHIP DIAGRAM FOR NORMALIZED RELATIONS



DATABASE CARDANILITY

USER TABLE: The Relationship in and from the following table is One-To-Many as User_ID is necessary in Buyer as well as Seller Table.

PRODUCT TABLE: The Relationship in and from the following table is Many-To-Many as Product_id is being referenced in Transactions Table and Messages table. And Category_id is a Foreign Key.

BUYER TABLE: The Relationship in and from the following table is One-To-Many as Buyer_id is being referenced in Transactions Table and Messages table.

SELLER TABLE: The Relationship in and from table is One-To-Many as Seller_id is being referenced in Transactions Table, Messages table, Product Table, User Table. Therefore, the cardinality of their relationship is ONE-TO-MANY.

CATEGORY TABLE: The Relationship in and from the following table is One-To-Many as Category_id is included in Feature Table and Product Table.

FEATURE TABLE: The Relationship in and from this table is One-to-One as Feature_id is being referenced in Product Table.

TRANSACTION TABLE: The Relationship in and from this table is Many-to-Many as Product_id is a Foreign Key of Product Table, Buyer_id of Buyer Table and Seller_id is of the Seller Table.

MESSAGES TABLE: The Relationship in and from this table is Many-to-Many as Product_id is a Foreign Key of Product Table, Buyer_id of Buyer Table and Seller_id is of the Seller Table.

1.d) Database of Online Advert system that consists tables such as:

Buyer Table: *PRIMARY KEY* (**Buyer_id**)

User_ID(F.K)	Buyer_id (P.K)	Buyer_Name
1010101	123456	Shazibali

Seller Table: *PRIMARY KEY* (**Seller_id**)

User_ID(F.K)	Seller_id (P.K)	Seller_Name
1010101	7616222	Express Guys

Product Table: *PRIMARY KEY* (**Product_id**)

Category_id(F.K)	Feature_id(F.K)	Product_id (P.K)	Seller_id (F.K)	Name	Type	Price (in $)
124567	129299001	1890002	7616222	IPhone	New	2500

Transaction Table: *PRIMARY KEY (***T_id)**

T_id (P.K)	Date/Time	Product_id	Name	Seller_id	Buyer_id
17628800	2016-01-19 03:14:07	1890002	IPhone	7616222	123456

Product_id: Foreign Key with References to *Buyer Table->Buyer_id*

Seller_id: Foreign Key with references to *Seller Table ->Seller_id.*

Buyer_id: Foreign Key with References to *Buyer table->Buyer_id.*

SQL code for selecting names of buyers and sellers of a product:

```
SELECT t.Name, s.Seller_Name, b.Buyer_Name from Transaction t JOIN Buyer b ON
t.Buyer_id=b.Buyer_id JOIN Seller s ON t.Seller_id=s.Seller_id where
Product_id=1890002;
```

OUTPUT

P_Name	Buyer_Name	Seller_Name
IPhone	Shazibali	Express Guys

2.

a.

MongoDB is a schema-less database that can store multidimensional array of documents, MongoDB structures data into collections of JSON documents.

JSON documents are particularly useful for data management for several reasons.

- Accessing different fields in a relational database is a sophisticated job and hence in the case of online advert system we can say that the fields like User_id, Buyer_id and Seller_id are related with most of the fields so therefore a JSON document helps because it is composed of a set of fields which are themselves key-value pairs. This means each JSON document carries its own human readable schema design with it wherever it goes, allowing the documents to easily move between database and client applications without losing their meaning.

- In an online advert system, we have many relational tables that have their unique flexible data structures therefore MongoDB supports a richer and more flexible data structure than tables made up of columns and rows. JSON is a natural data format for use in the application layer

- In addition to supporting field types like number, string, Boolean, etc., JSON fields can be arrays or nested sub-objects. This means we can represent a set of sophisticated relations which are a closer representation of the objects our applications work with.
  In the online advert scenario, this is helpful in representation of the relations between the categories and their different features.

- In an online advert system, we need to map data for different case scenarios a relational database such as MySQL uses object relational mapping to map the objects of the database and the application which is very time consuming for and resource eating thus using JSON documents in our database means we don’t need an object relational mapper between our database and the applications it serves. We can persist our data in the right form for our application.

b.

USE CASES FOR DATABASE IN TERMS OF USER ACTIONS TO ACCESS DATA.

1. **USE CASE-1**

- **Description**: A User wants to know a few similar products to the product he has selected.
- **Prevalence**: A common search
- **Suitable Aggregation**: Document structure is one document per product hence we can't just keep all products in one document.
- **Suitable Sharding**: Keeping all products in a category on the same shard.

2. **USE CASE-2**

- **Description**: A user knows the category of the product, but needs help choosing
- **Prevalence**: A common search
- **Suitable Aggregation**: Keeping all products in a category in one document is not practical
- **Suitable Sharding**: Keeping all products in a category on the same shard.

c. **People, Objects and classifications(Categories).**

All these Aggregates depend on each other as they are linked to each other and can evolve in the future, so basically Objects need to be sorted into categories for easy classification and to reduce data redundancy and to build a robust application.

People.

- **Meaning**: People is a potential aggregate in the online advert system. They are users who control the how they want the system to respond to them. Users ca interact with other users on the same system via message passing systems thus ensuring trust and satisfaction before thinking of purchasing a product.

- **For the Online Advert System**: The suggested aggregate people is the most dependent aggregate in the online advert system and without the people the online advert system would not work. All the interactions that are carried out in the database and done by the people. In case we see the above use case 2 the user must search to get the suggested category. Users actions are necessary for the database to respond in different manner for each case.

- **Against the Online Advert System**: People are users and users are needed to make the online advert system active, without the people the online advert system cannot function because with their actions the whole application functionality works.

Objects.

- **Meaning:** Objects are real time entities that are displayed in the online advert system. These objects have unique features and belong to a specific category. Objects can be sold or bought in this system. Users can interact with each other regarding a product they want to buy or sell.

- **For the Online Advert System:** This suggested aggregate object is necessary in an online advert system as it is needed for the system to work. Objects are potential aggregates in this system. Objects help the application in making it more classified and clear for a user about the system and what the user can see on the application. It helps the people in making clear decisions on the purchase. So basically, for an online advert system objects gives the initial functionality the system needs.

- **Against the Online Advert System:** Objects are aggregates that when used in such an application they need to be structured in such a way that whenever a user needs to access any information about them it needs to be structured in a way the functionality remains consistent. Not using Objects in this application will make it dead as it is a necessary entity that is needed in this system.

Classifications:

- **Meaning:** Classifications are categories in which many products are present. For example: A Product Skoda Laura is in the category Motor Cars and has a feature i.e. Model known as Skoda Laura

- **For the Online Advert System:** Classifications help the programmers to structure the database in a way that the functionality and efficiency of the application remains consistent and the application is robust.

- **Against the Online Advert System:** In an online advert system if the database is not classified and categorized then basically it makes the database have numerous number or anomalies and makes the database structure fail and the database will not withstand the load and will collapse.

d.

Database schema is the data structure of a database table, in RDBMS, such as MySQL, every database table should have a fixed data structure. Schemaless means the database don't have fixed data structure, such as MongoDB, it has JSON-style data store, you can change the data structure as you wish. With a schemaless database, 90% of the time adjustments to the database become transparent and automatic.

Restrictions.

Restrictions on Field Names.

Field names cannot contain dots (i.e..) or null characters, and they must not start with a dollar sign (i.e. $).

Restrictions on Fields (Product_id, Seller_id, buyer_id, Category_id, Feature_id, User_id)

These fields cannot use alphabets and only should be in numeric value.

Cannot contain Underscore (_), Comma (,), or any other special characters.

Advantages:

It helps is structured the keys in an efficient manner and there are less chances of typing errors if these restrictions are hardcoded. Thus, all these fields can have only numbers ranging from 1-to-n.

Restrictions on products Price only in Dollars

This field (Price) can only be entered and the currency is dollars and it cannot be implemented in another currency. So, we embed Dollars in the Price Document and make it a single unique currency in which the sellers can enter the products.

Advantages:

It makes the document much clear since the object is having only a single currency in which the products can be registered on the website.

It also makes it easier for the programmers to calculate the metadata of the database collection.

e.

Indexes are used to quickly locate data without having to search every row in the database collection.

Let us consider the entries for the sale of an object.

Possible fields in such a case are:

Brand, Rating, Price, Time

I need to find all products per category, and sort it by rating, then by price, but the final user may also be wanting to just sort it by price directly.

I create 3 indexes: **{brand :1}, {rating:1}, {price:1}, {time:1}**

These are the Following queries I can perform with the following indexes.

Here Brand is sorted in ascending order showing alphabetically from A-to-Z or a-to-z.

db.Advert.find().sort({brand:1})

Here ascending is sorted in descending order to show the highest rated product first.

db.Advert.find().sort({rating:-1})

Here Price is sorted in ascending order thus showing lowest price first.

db.Advert.find().sort({price:1})

Here time is sorted in descending order as to see the most newly added products.

Db.Advert.find().sort({time:-1})

- Indexing increases the response time for a query and makes it quicker.
- It is helpful for a large collection of records because it query runs very fast.
- Indexing every field is not helpful as we have to think about how the end users search on the online advert website.

For example a user searches Apple IPhone with the lowest price and highest rating.

Indexing all fields is not a good option because every field will have unique data and indexing on common fields is required as we want the results to appear quickly as the query runs fast on the database collection.

f.

Sharding is a term used for partitioning data across servers to enable:

- **Geo-Locality**: to support the geographical distributed deployments to support optimal UX for customers across vast geographies.
- **Scale**: needed by modern applications to support massive workloads and data volume.
- **Hardware Optimizations**: This is basically on performance versus Cost.
- **Support Lower Recovery Times**

Sharding includes a shard key that is assigned as a data modeler to model the partition of the data set.

Data is partitioned into chunks of data by the shard key and are further stored across many different physical servers.

MongoDB has three types of sharding: **Ranged, Hashed and Tag-Aware**.

For this case, we will be using the Ranged type of Sharding.

The Shard key for the Online Advert system would be:

Shard Key: {User_Id}

In the Online Advert System Application, it is designed to give every user that registers on the website a unique User_Id and then the user can decide their user_name and password.

For Example, we have 1000 User_Id's,

In ranged sharding we partition the data by the shard key in different ranges stored across many servers.

Chunk 1	Chunk 2	Chunk 3	Chunk 4	Chunk 5	Chunk 6
(-Infinite Numbers) - 200	**200 – 400**	**400 – 600**	**600 – 800**	**800 - 1000**	**1000 – (+*Infinite numbers)***

- It is more relevant to use a shard key because in the case of an online website that contains a million users these users will generate a lot of load on the database and thus it may crash or some fault if by chance is occurred in the database can be resolved by using sharding.

g.

METHOD 1

In this method, we specify the sender and recipient of the messages is by treating each message like a “blog” post.

- When the message is created then we add two users into the array. (Initially it doesn’t matter who is the sender/receiver)
- The response now would be treated as a comment and hence it is inserted into an array.
- The user that comments first is the sender of the message and the person to comment second is the recipient of the message.
- The recipient then replying to the message then is the sender and the sender of the message that receives a reply is then the recipient of the message.
- The sequence is in a loop till the end of the conversation.

For Example: User1: Kevin and User2: Shazib
objectID is the instance to each message created.

Then the values kevin and shazib are added into the array.

Kevin starts the conversation and comments first therefore kevin is the sender and shazib is the recipient of the first message

Thus, shazib replies to kevins message and thus become the sender and kevin become the recipient of the second message.

Example CODE: {

```
"_id" : <objectID>,
"users" : ["kevin", "shazib"],
"user_msgs" :
   [
      {
         "is_sender" : "kevin",
         "msg_body" : "Hi shazib, how is the assignment going on?!",
         "timestamp" : <generated by Mongo>
      }
      {
         "is_sender" : "shazib",
         "msg_body" : "All good Sir ! I am on the last sub question of the second question?!",
         "timestamp" : <generated by Mongo>
      }
   ]
}
```

METHOD 2

In this Method, we can know the sender and the recipient of the message in MongoDB is by having sender_id and recipient_id fields in the Messages collection.

The sender_id and recipient_id fields would just hold value for the appropriate user.

It also has the instance of the Object_id or Product_id.

Sender_id is allocated to the User who starts the conversation and recipient_id is automatically allocated to the User to whom the Sender is sending a message.

For Example:

There are two users on an online advert website.
USER1-
USER2-

If USER1 is a Buyer of a product and thus contact the Seller of that product, then USER1 is assigned a new Sender_id in the Message Collection.

USER2 is the recipient of the product and hence USER2 is given a unique Recipient_id.

The sender_id and recipient_id is assigned to the users till the end of the conversation.

If any of the users end the conversation and start a new conversation, then the sender_id and recipient_id will be re initialized

Sorting of the Messages can be done by introducing an additional field Timestamp in the collection.

I would choose this method for my application.

3.

a. List the prices (and no other variables, including the ID) of all cars for sale where the make is Skoda and Model is Octavia.

QUERY:
mongos>
db. Assgcars.find(Manufacturer:"Skoda",Model:"Octavia"},{Price:1,_id:0})

OUTPUT:

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

b. Calculate the average price of all Skoda cars for sale.

QUERY:
mongos>db.Assgcars.aggregate([{$match":{"Manufacturer":"Skoda"}},{"$group":{"_id":"$Manufacturer","AVGSKODA":{"$avg":"$Price"}}}])

OUTPUT:

```
mongos> db.Assgcars.aggregate([{$match:{"Manufacturer":"Skoda"}},{$group:{_id:"$Manufacturer",Avg:{"$avg":"$Price"}}}])
{ "_id" : "Skoda", "Avg" : 5853.97037037037 }
mongos>
```

c. Calculate the average price of each different model from Skoda cars for sale.

QUERY:
mongos>db.Assgcars.aggregate([$match:{"Manufacturer":"Skoda"}},{$group:{_id:"$Model",Avg:{"$avg":"$Price"}}}])

OUTPUT:

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

d. Find the model of Skoda car with the highest average price.

QUERY:
mongo>db.Assgcars.aggregate([{$match:{“Manufacturer”:”Skoda”}},{$group:{_id:”$Model”,Avg:{“$avg”:”$Price”}}},{$sort:{Avg:-1}},{$limit:1}])

OUTPUT:

```
mongos> db.Assgcars.aggregate([{$match:{"Manufacturer":"Skoda"}},{$group:{_id:"$Model",Avg:{"$avg":"$Price"}}},{$sort:
{Avg:-1}},{$limit:1}])
{ "_id" : "Yeti", "Avg" : 5945.356 }
mongos>
```

e. All cars have field called Extras which holds an array of strings (e.g.{“Extras”:[“ABS”,”PAS”,”Aircon”]}.Write code to list all of the extras ever mentioned for each manufacturer in turn (i.e. the union of the Extras arrays grouped by manufacturer).

QUERY:
mongo>db.Assgcars.aggregate([{$group”:{“_id”:”$Manufacturer”,”extras”:{“$push”:”$Extras”}}},{“$unwind”:”$extras”},{“$unwind”:”$extras”},{$group”:{“_id”:”$_id”,”ManufacturerExtras”:{“$addToSet”:”$extras”}}}])

OUTPUT:

```
mongos> db.Assgcars.aggregate([{"$group":{"_id":"$Manufacturer","extras":{"$push":"$Extras"}}}, {"$unwind":"$extras"},
 {"$unwind":"$extras"}, {"$group":{"_id":"$_id","ManufacturerExtras":{"$addToSet":"$extras"}}}, ])
{ "_id" : "Fiat", "ManufacturerExtras" : [ "ESP", "Auto Wipers", "Parking Sensors", "Aircon", "Power Windows", "PAS",
"ABS", "SatNav" ] }
{ "_id" : "VW", "ManufacturerExtras" : [ "SatNav", "ESP", "PAS", "Parking Sensors", "Power Windows", "Auto Wipers", "A
ircon", "ABS" ] }
{ "_id" : "Skoda", "ManufacturerExtras" : [ "ABS", "PAS", "ESP", "SatNav", "Parking Sensors", "Aircon", "Auto Wipers",
 "Power Windows" ] }
mongos>
```