UNIVERSITY of STIRLING

ITNPBD3 – Relational & Non-relational databases
Course Assignment

David Haveron – Student Number 2527317

Question 1

a.) If this classified advert website were to advertise, say guitars only for example, there would likely be a set of descriptive features or attributes which could be collected from the seller and stored in a single relation to capture data about the guitar - the type (electric/acoustic), the colour or material and condition (among other standard features used to describe guitars). This would, in some cases, be an adequate database design for a website selling similar items which can broadly be described by the same attributes (that is, selling guitars or selling mobile phones or selling motor bikes, for example).

However, as we wish to store many fields to describe a variety of classifications (guitars and mobile phones and motor bikes), in the database, this creates a problem as each classification has a set of different (often unique) features to describe their respective category, compared to a single set of features to describe a single classification of similar items only, as explained in the example above. Following this logic, the constraints of the relational model and normal form become more problematic for the reasons below:

- Some fields may contain several (or more) values - in the case of the guitar a field 'accessories' could contain a list of accessories such as guitar bag, amplifier, guitar strings, among others. Listing these accessories in a single field would mean the relation is not in first normal form.

- Some fields for the same product may be described by different features, that is, one seller of a guitar may wish to advertise the accessories included in the sale and perhaps the condition of the guitar, while another may wish to only specify the colour and the manufacturer – this results in an incomplete database (blanks or nulls) with poor integrity/consistency.

- A new product which comes to market may possess new fields or qualities which have not yet been included in the database and the database designer will have to go back and re-design the database to include these additional fields. This well may include the redesign the application so the application knows there is a new additional field as part of the descriptive features for that product, and, doing this update while the database is live and in use by the application introduces additional challenges.

- The relational model is structured in such a way that all the fields necessary for the database, need to be full-defined (data types, lengths – among other specifications) in order to fully specify and define the relational schema model. This means the database designer needs to carefully consider and anticipate which fields might be necessary for the database.

The two approaches to solving this problem are the 'one-table' approach and the 'two-table' approach which are described below:

One-table solution:

If one were to create a single relation (table) to capture all these sets of attributes for each classification (guitars, mobile phones, motor bikes), there would be entries or rows of data for which some of the descriptive fields may be relevant and other descriptive fields may be non-relevant (or not applicable, Null), as seen in Table 1 below. The assumption here is the relation has been show in First Normal Form (1NF) with one entry of data for a given row/column, as it is the first requirement for achieving a relational database. The fields corresponding to the items guitar, mobile phone and motor bike have been colour coded respectively.

Item_Number	Item	Type	Colour	Condition	Make	Model	Network	Mileage_miles	Service_history
1	Guitar	Electric	Red	Good	Fender	Strat_Artic	Null		Null
2	Mobile_phone	Samsung	Blue	Aged	Samsung	Galaxy	Vodaphone	Null	
3	Mobile_phone	Null	Black	New	Samsung	LTE	O2		
4	Motor_bike		Silver	Well_maintained	Triumph	Street_scrambler	Null	134,000	No

Table 1 - One table approach for classifying guitars, mobile phones and motor bikes

This relation in Table 1 contains some advantages and disadvantages as which have been described below

Advantages:

- Query performance is better (than the two-table approach) and latency is low as no joins are required – all the data about each product for sale is stored in a single relation. This is an advantage from the customer's perspective as queries are returned quickly.

- The development of joins for assorted queries across the database can be done without too much thought into which relations will be required in the query – so the joins necessary are perhaps less complex than the joins necessary in the two-table approach.

Disadvantages:

- Database integrity is poor as there could be empty fields or null values. This results from a particular attribute (or “descriptive field”) being relevant for describing one item for sale but not for another. This in turn introduces challenges when querying and retrieving data from the database.
- Could contain data redundancies/anomalies and duplication, so query results may be inconsistent or erroneous
- Additional storage required as data is duplicated and occasionally redundant in the relations
- Insertion and deletion of records become problematic and this could introduce data anomalies. Products, such as mobile phones for example, are always being developed to include new features/qualities which may not yet be in the database, and these become difficult to include in the relational database design
- Relational ACID transactions become problematic for writes with locking and permissions

Two-table solution:

Using the two-table approach the designer aims to decompose and further normalise a single relation (Table 1, for example) into separate relations and query the database through the use of joins. This normalisation process should be a non-loss decomposition (that is no data is lost in the process) and puts the database into an appropriate normal form (2NF or 3NF). The process of normalisation involves dividing the database into two or more tables (of grouped related entities in this case) and defining the relationships between them through a set of primary and secondary keys.

Advantages:

- Database integrity is protected as the database is well structured to store data more efficiently and facts are stored only once with little or no replication of data – resulting in a more accurate database which doesn’t require unnecessary storage.
- Complete and consistent database as the relations uniquely capture relevant data and link relationships between relations – a more ‘lean’ collection of data.
- These normalised relational database models are mature and work well – they have frequently used and have stood the test of time as a reliable model (for specific database applications).
- There are many people who have skills to work with the relational database models, meaning there is no shortage of proficient, inexpensive professionals to manage and run the databases.
- The ACID transactions and strict integrity mean they can be relied upon to be correct – this becomes important in certain applications (financial transactions, for instance).
- The database is designed around the data (meaning that many applications and programming languages should be able to use it, without putting too much consideration into the application itself).

Disadvantages:

- Additional joins are required, introducing complex joins which become difficult to manage for particular queries.
- More computational resources required by the database if user performance (low latency) is of importance which is an additional commercial overhead to the classified company.
- Additional database designer resources may be required to ensure entity and referential integrity are adequately achieved and maintained.
- Web applications which collect and contain large volumes of data are often suited to be distributed over a cluster for reduced latency and hence an improved user experience on a website. Website users are easily swayed to jump to another website if one particular classifieds website is taking an unusual amount of time to return search results. This becomes a commercial liability if you are losing customers due to latency as the user experience then becomes poor and user traffic will begin to decrease. And, although relational model can be optimised for speed if run over a cluster but this can be difficult and this introduces problems with integrity as licence costs can be restrictive over many machines.

From the two solutions, one table and two table approach, I would choose the two table approach for the numerous advantages it holds over the one table approach however more time would be necessary to fully develop, and specify fully, the relational database and the common (complex) queries which the classified web application would require.

It becomes clear many collections of data may not fit appropriately or naturally into a relational schema model and the constraints of normal form which are encouraged/enforced to protect database integrity and increase reliability, can be too restrictive for storage of the classifieds data in a relational database. In addition, the benefits of distributing a database over a cluster (which is economic and more efficient for large applications requiring low latency) mean an alternative database model is better suited to serve as a database for the classified advert website where the data is always available and latency is low as reasonably possible (opposed to absolute integrity which the relational model offers).

b.) For illustrative purposes and simplicity of the joins with relations, the normalised model shown below follows the one table approach/solution explained above. The Primary Keys are underlined and in bold and the Foreign keys are highlighted in yellow.

People						
User_id	Name	Surname	Address	Postcode	DOB	Email
1234	*John*	*Smith*	*Willow_Lane*	*SW64KA*	*27/07/1979*	*jsmith@gmail.com*
1235	*Robert*	*Brown*	*Cresent_Road*	*FK94HB*	*16/11/1989*	*rbrown@yahoo.com*
…	…	…	…	…	…	…

Relation Schema: 'People' ('User_id' , 'Name' , 'Surname' , 'Address' , 'Postcode' , 'DOB' , 'Email')

The 'User_id' will be an artificial and unique primary key within the 'People' relation, as values will be incremental generated or assigned as users register with the website. This methodology ensures the primary key will never be blank or null and will uniquely identify the row to which it has been assigned.

A candidate key could alternatively be 'Surname ', 'Postcode ', 'Email '(for example) as it is highly improbable to have two people with the same surname, postcode and email address, however an artificial key guarantees a unique primary key.

This 'People' relation is in Third Normal Form (3NF) as all attributes within a row of this relation are fully (functionally) dependant on the Primary key, 'User_id'.

Transactions					
Trans_no	Trans_date	Debit_Acc	Credit_Acc	Buyer_id	Product_id
1	*22/11/2016*	*123782137*	*172635262*	*1234*	*2342*
…	…	…	…	…	…

Relation Schema: 'Transactions' ('Trans_no ', 'Trans_date ' , 'Debit_Acc ' , 'Credit_Acc' , 'Buyer_id', 'Product_id')

The 'Trans_no' will be an artificial and unique primary key within the 'Transactions' relation, as values will be incremental generated or assigned as transactions occur (and are registered) through the website. This methodology ensures the primary key will never be blank or null and will uniquely identifies the row or entry for each transaction.

A candidate key could be the composite key consisting of 'debit_acc','credit_acc' and a timestamp of some sort as no two transactions between two accounts can occur simultaneously with the same timestamp, again it seems apt to use an artificial key to guarantee a unique primary key.

This 'Transactions' relation is in Third Normal Form (3NF) as all attributes within a row of this relation are fully (functionally) dependant on the Primary key, 'Trans_no'.

Messages				
Message_id	Receiver_id	Date_sent	Message	Sender_id
121212	*1234*	*13/10/2016*	*"Hello John Smith…"*	*1235*
…	…	…	…	…

Relation Schema: 'Messages' ('Message_id' , 'Receiver_id ' , 'Date_sent ' , 'Message ', 'Sender_id ')

The 'Message_id' will be an artificial and unique primary key within the 'Messages' relation, as values will be incremental generated or assigned as messages are created between users through the website. This methodology ensures the primary key will never be blank or null and will uniquely identifies the message entry recorded in the database.

The only other possible super key or candidate key would be a composite key consisting of all the fields in this relation, so it seems plausible to introduce an artificial primary key for each new message sent through the website.

This 'Messages relation is in Third Normal Form (3NF) as all attributes within a row of this relation are fully (functionally) dependant on the Primary key.

Products					
Product_id	Seller_id	Manufacturer	Model	Classification	Price_gbp
2342	*1235*	*Samsung*	*Galaxy*	*2*	*50*
3142	*1234*	*Triumph*	*22*	*3*	*500*
4223	*1234*	*Gibson*	*EF*	*1*	*400*
...	...	...	...	...	...

Relation Schema: 'Products' ('Product_id ' , 'Seller_id ', 'Manufacturer '' , 'Model , 'Classification' , 'Price_gbp')

The primary key for any product listed in the 'Products' relation, will be the artificial key 'Product_id'. These primary keys will be unique, as values will be incremental generated or assigned as the product is listed on the website and hence created on the database. This methodology ensures the primary key will never be blank or null and will uniquely identifies each listed product registered for sale on the database.

This 'Products' relation is in second normal form as the relation could potentially be decomposed into separate relations.

Classifications	
Classification_name	**Classification_number**
Electric_guitars	*1*
Mobile_phones	*2*
Motor_bikes	*3*
...	...

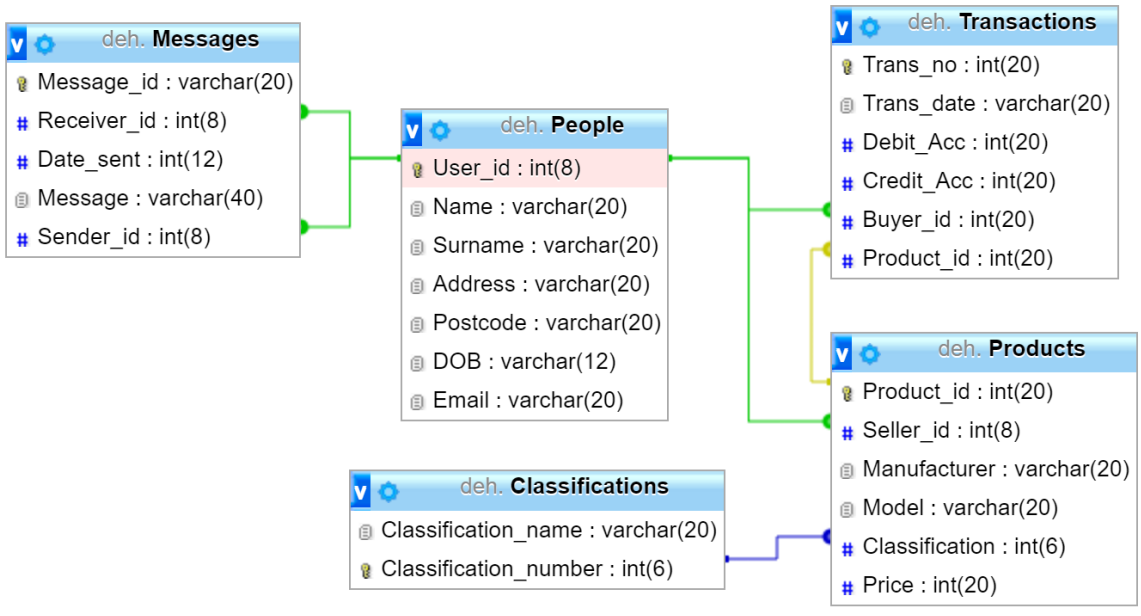
Relation Schema:

'Products' ('Classification_name ' , 'Classification_number ')

This 'Classifications' relation is in third normal form (3NF) and cannot be further decomposed.

Largely it could be said that time should be spent identifying from the super keys, those which are appropriate candidate keys and ultimately from those, which are best suited as a primary key given the context of the data in the relation to the real world (for example its highly improbable to have two John Smiths living at the same address), however for a website selling potentially millions of items over a given period and a database which might need to be amended over time, it seems plausible with this large volume of data to generate artificial primary keys to ensure individual people, transactions, messages and products are uniquely and correctly identified.

c.) This ER diagram has been developed using the 'Designer' feature in phpMyAdmin, where the relations have been specified, primary keys and secondary keys assigned and indexes allocated to create the relationships among the relations. The relationships show the one to many cardinalities throughout the relational model.



d.) The example code shown below queries a transaction or purchase which occurred between John and Robert, where John purchased a product (Samsung Galaxy) from Robert. The join was implemented knowing the Product_id (Product_id = 2342):

User_id	Name	Surname	Address	Postcode	DOB	Email
1234	John	Smith	Willow_Lane	SW64KA	27/07/1979	jsmith@gmail.com
1235	Robert	Brown	Cresent_Road	FK94HB	16/11/1989	rbrown@yahoo.com

Trans_no	Trans_date	Debit_Acc	Credit_Acc	Buyer_id	Product_id
1	22/11/2016	123782137	172635262	1234	2342

Product_id	Seller_id	Manufacturer	Model	Classification	Price
2342	1235	Samsung	Galaxy	2	50
3142	1234	Triumph	X2	3	500
4223	1234	Gibson	EF	1	400

```
SELECT
seller.Name AS Seller_Name,
seller.Surname AS Seller_Surname,
buyer.Name AS Buyer_Name,
buyer.Surname AS Buyer_Surname
FROM `People` AS seller

JOIN `Products` ON seller.User_id=Products.Seller_id
JOIN `Transactions` ON Products.Product_id=Transactions.Product_id
JOIN `People` AS buyer ON Transactions.Buyer_id=buyer.User_id

WHERE Products.Product_id=2342;
```

Shell output:

Seller_Name	Seller_Surname	Buyer_Name	Buyer_Surname
Robert	Brown	John	Smith

Question 2

a.) The challenges we encountered using a relational database for a classifieds online business were due to the vast variety of fields to describe 'items for sale' in a single database. Using the first approach of putting these fields all into a single relation would introduce some challenges or disadvantages for the relational database design. And, although the second approach achieved a better design than the single relation approach, there were still some difficulties and limitations identified as the constraints of the relational model can be too restrictive (on top of performance challenges). In addition, it was noted that a relational database is not easily distributed over a cluster (putting the entire database on a single machine, leaves the database vulnerable as a single point of failure which could be catastrophic for a commercial web operation). And, for large volumes of data and web applications, a cluster distribution is a preferential for a reduced latency (improved user experience), replication/sharding and map reduce operations.

MongoDB solves many of these challenges and is well suited for a classified advert application, as it is capable of storing many different types of objects with different sets of attributes. A NoSQL solution also removes a number of the restrictions we encountered with the relational database model. MongoDB databases are designed with the specific application or task in mind and offers a host of benefits such as flexible schema (schema less) where the collections do not enforce document structure, and, it more easily or efficiently distributed over a cluster when the database becomes large. In addition, because of this distribution of the database over clusters, it lends itself to map reduce operations.

For the classified advert application, the requirements of availability and low latency are of greater importance (for improved user experience) than the absolute integrity which the relational model offers. With an increased popularity of agile programming methods, it seems appropriate to use a database which is easy to alter during rapid growth and evolution of a database.

In summary, choosing the NoSQL MongoDB as a supporting database for the classifieds website, provides a host of advantages which solve many of the problems encountered with the relational model and offers additional functional benefits:

- Data in MongoDB has a flexible schema. The data is also stored in BSON format which is advantageous as data types can vary and only data which is necessary or relevant can be stored (null values need not be stored).
- The schema and content and length from one document (describing guitars for sale, for example) could contain entirely different schema to another document (describing mobile phones, for example).
- As no complex joins are required the database provides reduced latency resulting in faster queries for the user, provided the database is appropriately aggregated, indexed and sharded.
- The MongoDB database scales far more efficiently than a relational database and lends itself to inexpensive storage if the database is sharded and indexed correctly.
- MongoDB supports dynamic queries on documents using a document-based query language, which means many SQL queries (if the database was originally a relational one) translate easily to MongoDB document-based query language.
- There is a shift towards object orientated programming which means there is an object relational impedance mismatch. To further clarify, there is a trend towards object orientated programming for numerous advantageous functional reasons however this object orientated programming is far less efficient and compatible with a relational database design. An element of this incompatibility of the relational model with object oriented programming is due to the fact that rich objects have to be normalised to store in a relational database and converted back to objects during read processes, whereas no conversion is necessary with MongoDB as collections of documents already contain objects.
- MongoDB allows indexing on any attribute resulting in quicker queries.
- Unique object or document keys are automatically assigned to each document which offers functional indexing/querying advantages
- It becomes far easier to add and remove (or update) attributes without the strict schema enforced or imposed by the relational database
- A relational database cannot be easily distributed over a cluster, where as a MongoDB database is efficiently distributed and replicated across a cluster lending itself to replication for high availability or auto sharding (which can be useful for efficient sharding of databases as data is spread and maintained in an optimal way for even distribution across all the nodes)
- MongoDB provides easy creation, insertion, read, deletion, update/extend for documents or embedded documents so functionally it becomes easier to work with as a database
- MongoDB supports the Map Reduce model on clusters which allows for large scale distributed queries on with the database

MongoDB has become synonymous for Big Data applications for many of the reasons explained above, and, it follows it would serve well as a suitable database to support the application of a classified advertisement website.

b.) User case One:

The user decides he/she wants to buy a product (an electric guitar, for instance) on the classifieds advert website. The user will begin by logging into their account on the classifieds website and proceed to search for a particular product (query the database to return all the products on record as 'iPhone 6s', for example). The database then returns a list of entries listed from the database which match the query for 'iPhone 6s'.

Ideally if the database is designed efficiently, the database will return only a few important key features for each record/product listing (a summary perhaps of each listed product and document) such as the product cost, a brief description and perhaps a

picture of the product. Should the user then wish to find out further information about a particular product the user will click on the specific product of interest and this will open (or return) the full content of that particular record/document for the user to review.

User case Two:

The user has already decided to buy a product from a particular seller and in addition decides to search for all products that seller is selling (perhaps the user has bought a couch and wishes to buy additional items to furnish a new flat). As a collection trip has already been planned to collect one item, it seems feasible to purchase any other suitable items from the same seller which can be collected in a single trip, opposed to different sellers in different locations requiring multiple trips.

For this example, the user will navigate the website to the seller's profile which will have a list of entries or records for all the products or items for sale by that seller (query the database to return all documents advertised for sale by that particular seller). Practically an example of how this is executed would be that the buyer clicks on the product which they have purchased. On that page there is a field or link for the seller profile, and the buyer would click on the seller profile. This would bring up an additional document/web page where the buyer can request from the database, a list of all the items that particular seller has submitted. The database will return all those individual items for sale which the seller has submitted to the database and the buyer can now decide to buy additional items from the seller (assuming the seller has additional items listed in the database).

c.) Aggregate data models are the NoSQL way of grouping or organising (useful) units of data in different (application appropriate) ways in order to ensure the most frequent/common use case is served the fastest. In the relational model, the relations for the classified advert design would be reviewers, products or messages as these appeared initially to be the most useful approach to group/gather data into relations with similar attributes. Similarly, when designing aggregations (groups) for a document database the designer will need to decide what data goes into the objects, how big the objects are and what fields are necessary in the objects which most functionally and efficiently serves the application.

People

{“ID”: 837263,

“User id” : 123

“User name” : John Smith

“Product”: [

{“Product”: “iPhone”,
“Model”: “6s”, “Colour” :
“White”, “Sold” : 1},

{“ Product”: “Mountain Bike”,
“Manufacturer”: “Giant”,
“Model”: “Avalanche”,
“Colour” : “Blue/Black”,
“Sold” : 0},

{“Product”: “Blender”,
“Manufacturer”: “Kenwood”,
“Colour” : “Blue”, “Sold” : 0},

]}

An illustration of people (User product listings) as a potential aggregate, can be seen to the left where the aggregation groups all the records or objects which any given user has listed on the website, under the users name or user identification. The aggregations would consist of a list of similar structure documents, where each document contains the name of the user (selling or listing products on the database) and inside that document a set of products for sale.

The consideration to be made is about the use cases and the question to ask is whether a query by a user on the website would require fast access to all the products for sale by a given user? This does tie in quite well with the use case two above where a particular buyer may have already decided to buy a couch from a particular seller, and decides to browse other listed products for sale by that seller in order to furnish a flat.

Objects

{“ID”: 283728,

“Name” : “Apple iPhone 6s”

“Objects”: [

{ “Model” : “6s”, “Colour” : “Black”, “Condition” : “Good” , “Network” : “02”, “Price” : “£80” },

{ “Model” :” 6s”, “Colour” : “Blue”, “Price” : “£100 negotiable” },

{ “Model” :”6s”, “Colour ” : “Silver”, “Condition” : “New” , “Price” : “£120”, “Charger”: “no”},

]}

Aggregating the data by objects (or products for sale) would has been shown schematically on the left.

Similarly, the question being asked is whether this aggregation would serve a common or frequent use case?

Use case one described a potential use case where a user would search the website for a particular product, ‘Apple iPhone’ for example.

The aggregation of objects does seem plausible as it serves the use case one well, where a user could search for all the products (‘Apple iPhone 6s’) for sale on a website which would return this document containing many objects about each ‘Apple iPhone 6s’ listed on the website database. Ideally then the user could then sort and refine these results, according to cost or location for instance.

It should be noted users are more inclined to browse around between products which have similar functionality (see aggregation of classifications, below), opposed to being set on a particular product model and make, going directly into a website to search for “Apple iPhone 6s” and only purchasing one from those listed without at least first browsing alternatives or alternative categories.

Classifications Data

{“ID”: 273648,

“Classifications”: “Appliances”

“Appliances”: [

{“Product”: “Washing machine”, “Manufacturer” : “Whirlpool”, “Colour” : “White”, “Condition”: “New”}

{“Product”: “Toaster”, “Manufacturer” : “Kenwood”, “Model” : “”X200 “}

{“Product”: “Kettle”, “Manufacturer” : “Hobson”, “Colour” : “Red”}

]}

Finally looking at classifications as a potential aggregate, one could imagine the aggregate document (left) where each classification has list of items/objects which have been listed under that classification.

This could potentially serve as the most effective aggregate as users on product websites have generally developed a ‘browsing’ style where the user has some idea of the preferred product but is entirely happy to consider alternatives (and ultimately narrowing down a search), for which an aggregate of classifications would serve the user best and encourage a ‘browsing’ style of shopping and potentially identifying additional products of interest.

d.) With a relational database, there is high level of protection enforced by the strict schema. This means that if a field is specified as a numeric value and the user tries to input a string, the database will reject the string in that field, and only accept numeric values.

The MongoDB definition of a schema less design describes that collections of documents may have documents which vary in the number of fields, content and size. Alternatively put, it is a database design in which the schema defining how the data is stored, is relaxed. With MongoDB as a document database, there are no pre-defined restrictions on what type of data is put into each field (key, value pair) and documents within the database have a dynamic schema. This means the data in the documents or collections need not have the same fields or structure. In addition, a field value in one document may have one type of numeric data and the same field in another document may contain a different type of data. The schemaless system creates a more flexible approach to redefine or update data within the documents.

MongoDB imposes a size limit on a single document (depending, so there is opportunity to store many (potentially millions) of entries with descriptive information about objects for sale. This seems reasonable then to limit or restrict the number of objects described in a single document to ensure there is even and functional distribution across the cluster of nodes. Or alternatively and perhaps more functionally one could design the application to only display a limited number of documents at a time, when returning the results from a user query and should the user wish to see the next set of results, the application returns the next set for the user to view. This process of limiting returned results can be seen in many product websites, as it seems appropriate to return a limited set of results to the user opposed to returning an entire collection of documents which might over-strain the system resources involved.

Finally, as one would expect a classified advert website to have pictures of the product for sale, users would be advised to upload photos of the product for sale and because of the size limit of a single document in MongoDB, it seems appropriate to ensure a size limit is set at the application level to only accept images under a specified size.

e.) The idea behind indexing in MongoDB is to support the efficient execution of queries. That is, indexing is advantageous as it supports our goal of providing a quick and efficient query process for the users, which improves user experience. Without any indexes MondoDB must scan every document in a collection to select those documents that match the query statement and these indexes are defined at the collection level (also supported on any field or sub field of the documents in a MongoDB collection).

Using indexing for our application means we would gain a more efficient search processes in comparison to searching every field, as, searching every field requires a write/update to the document which consumes unnecessary memory. In addition, the results will pre returned in a sorted format negating the need for an additional sort operation after a query. These index results don't need access to the database at all, the data is returned from the index.

The most perhaps feasible fields to index on would be the Product Name and Price fields as these could be said to be common searches a user would execute. That is, a user would typically search for a particular product, iPhone 6s, and those results returned would be of interest to the user and should be returned to the user in the shortest possible time. Similarly, Price as an index seems plausible as the product price is often a driving factor for a consumer making a final decision on purchase of a product.

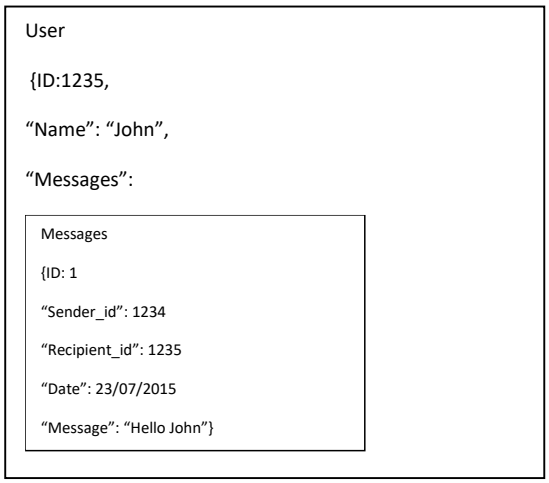
A combination of a compound index, Name and Price (in this order) should too be considered as these are perhaps the most likely attributes to be grouped by a consumer to specify/commence the search process for a product.

f.) Traditionally, databases were stored on one single machine with potentially a backup checkpoint or log for resilience. The approach of putting the database on a single machine, leaves the database vulnerable as there is a single point of failure if that machine failures or if the storage or network encounters problems. In addition, scaling the database requires new machines of higher storage capacity, which comes at a cost.

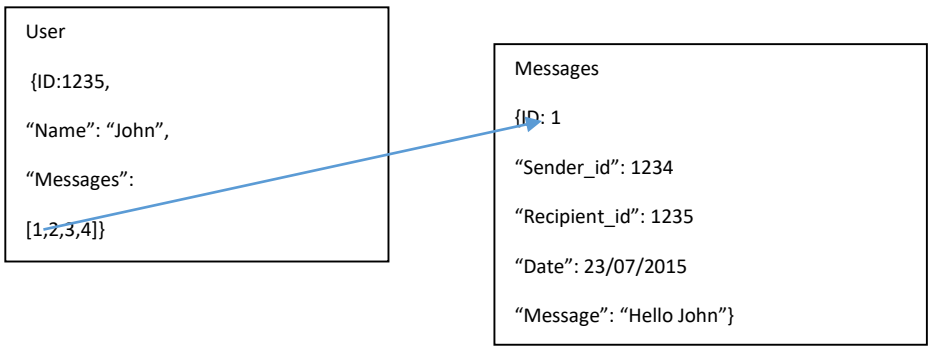
The alternative and potentially advantageous approach is to store the database over a cluster using either replication or sharding. Sharding is the process of spreading or splitting the data across each node in the cluster so that each node gets a subset of the data (across multiple machines). This is MongoDB's approach to accommodate databases which experience rapid growth through horizontal scaling. sharding of data can be done in with geographical location in mind (user location, for example) or sharded according to particular aspects or qualities of the data (products with similar prices are put into common machines) and this is often driven through use cases that are most frequent and where query time should be optimal for the user. The benefits of sharding the database include a quicker query process and a database less prone to failure due to the distributed nature of the data. In addition, as the database grows or reduces in size, this can be easily done by adding more machines to support the growth and demands of read/write operations or remove machines to create a more 'lean' environment. A shard key can be compared to an index and index page in a book, which allows a reader who has a particular topic or chapter in mind, to more effectively find or jump to the page of interest, in a quick targeted way - opposed to searching the entire book until the page of interest appears. Here it becomes clear that without a shard key, all the documents in the database will need to be queried until the document of relevance/interest is found - which introduces latency and increases query time. In addition, shard keys are stored in memory so this plays well to a reduced latency and the keys and documents need not be read from the database.

A shard key should ideally be based on one or more chosen fields which most frequent queries are likely to include. It seems plausible to use a shard key 'Product' (iPhone 6s) if classification was used as an aggregate as this serves as the most common query any given user will search by on the database. This shard key will also appear in every record and will have a large variety of values (which means it can be sharded across many nodes)

g.) One approach to perform server side joins in MongoDB is using document embedding. The idea behind document embedding is to serve a common query efficiently by aggregating the data together and embedding all the necessary data to serve that query, into a single document (as to reduce the number of queries required). For managing the data about messages sent and received through the website application, it seems sensible to aggregate or embed the whole conversation about between users and all the data about the conversation, into one document. The disadvantage with document embedding is that the embedded documents do not automatically update, and manual, multiple updates are necessary to maintain consistency within the database. This means that if documents require changes and multiple updates often in many locations, then latency is introduced with the write process and an alternative more automated solution may be feasible.



Manual References is the alternative solution to document embedding where documents are connected or joined through a reference identification number/value or pointer (this reference can also be an array).



Of the two solutions explained, and the context of specifying the sender and recipient of a message, the embedded document approach would serve as a solution as these messages, once sent are historic and are unlikely to be modified, updated or changed. If the information being stored was something like price, it would require an alternative to document embedding for autonomous and consistent update operations.

Question 3

a.)

```
db.Cars.find (
{ "Manufacturer" : "Skoda" , "Model" : "Octavia" },
{ "Price" : 1 , "_id" : 0 }
)
```

Shell output:

```
{ "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 }
```

b.)

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

Shell output:

```
{ "_id" : "Skoda", "avgPrice" : 5853.97037037037 }
```

c.)

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

Shell output:

```
{ "_id" : "Octavia", "avgPrice" : 5821 }
{ "_id" : "Superb", "avgPrice" : 5940.006666666667 }
{ "_id" : "Fabia", "avgPrice" : 5287.3 }
{ "_id" : "Yeti", "avgPrice" : 5945.356 }
```

d.)

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

Shell output:

```
{ "_id" : "Yeti", "avgPrice" : 5945.356 }
```

e.)

```
db.Cars.aggregate ([
        {  $group : { _id : "$Manufacturer" , "extras" :  { $push : "$Extras" }
                }},
        { "$unwind" : "$extras" },
        { "$unwind" : "$extras" },
        { $group:      {  _id:"$_id" , "extra" : { $addToSet : "$extras" }
                }} ])
```

Shell output:

```
{ "_id" : "Fiat", "extra" : [ "Auto Wipers", "Power Windows", "SatNav", "ABS", "ESP", "Aircon", "PAS", "Parking Sensors" ] }
{ "_id" : "Skoda", "extra" : [ "SatNav", "Auto Wipers", "Power Windows", "PAS", "ESP", "Aircon", "Parking Sensors", "ABS" ] }
{ "_id" : "VW", "extra" : [ "ABS", "ESP", "PAS", "Aircon", "SatNav", "Power Windows", "Auto Wipers", "Parking Sensors" ] }
```