1.

The website I have chosen for this assignment is one of the most popular social media websites used globally, Twitter (1). For this assignment we will assume that Twitter uses a relational model and therefore the proposal will explain why a NoSQL model should be implemented. I have chosen Twitter for this assignment as it is a clear website that I believe should implement a NoSQL solution due to its specific characteristics which will be explained further in this assignment. Furthermore, Twitter’s recent acquisition by Elon Musk sets a realistic scene for this assignment to follow as I will write my proposal to influence database decisions Elon Musk would potentially want to implement on Twitter.

Twitter is a well-established social media website that has been around for over fifteen years and has a huge user base estimated to be at 290.5 million users (2). Twitter focuses on one main feed that shows the posts of individuals, companies, news outlets and other media entities that you may follow, there is also a trending tab which acts as a feed for current news topics regarding live events going on around the world. Twitter can also be used as a direct messaging service. These are the core features of the Twitter platform. Twitter differentiates from other social media platforms as it is largely focused on the most up to date events rather than sites such as Instagram and Facebook which arguably act as more of an online personal diary.

Twitter needs to store a huge collection of different types of data for all its different features. Twitters largest data collection source would come from storing each individual tweet, each tweet allows for two hundred and eighty characters which needs to be stored, along with this data Twitter must store the user who posted it, the time, location and attached information on likes, retweets and comments from users interacting with said tweets. Twitter must store every user’s basic information such as their login details, name and email, these values must not be left null as they are used to identify each unique user. Twitter also stores a wealth of additional data which is not as frequently interacted with such as your individual preferences, display settings, security, languages, and notifications to name a few. Twitter will also store direct messaging related data; this data must have the highest level of security due to the nature of protecting personal conversations and therefore be encrypted.

Twitters database can be broken down into different entities and their respective attributes. The scope of Twitters database will of course be extremely large therefore the core tables / documents that Twitter functions around will be focused on in this assignment. The most essential entity will be the user, this consists of attributes such as username, password, and email address, these three values cannot be null. Furthermore, the user entity can have attributes such as birthdate and financial information. Twitter would also need an entity regarding tweets, the attributes for this would be the date, time, location, text, image or video, Twitter would also need a several entities relating to this such as a retweet entity, likes entity, and a comments entity, all these entities would draw from links between different user id’s interacting with different tweet ids in different capacities. Furthermore, Twitter would also need a messaging entity with different user id’s interacting with other user id’s the attributes of this would consist of text, images, videos, date, time etc. Twitters trending tab would require many different entities for each rank in the trend, this would be accompanied by many different attributes to display the relevant information per trending topic.

The scale of entities and attributes in Twitters entirety is extremely large and detail of each one would simply require a novel of details, the entities and attributes outlined in this assignment are the ones that require the highest frequency of data interaction whereas many of the other features available on Twitter are not used as regularly such as updating your security details which has relatively low traffic compared to the core features. Depending on what database design is implemented will determine the entities and attributes that belong to it, in a relational model there will be many different entities which brake down the data needed to be stored into different normal forms, under NoSQL many attributes can belong to the same entity so that all the data is stored together in the same place. We must also acknowledge that in a relational model using MySQL we have to define the data types for each of the attributes, this would consist of varchars, ints, date/times etc, whereas with NoSQL these data types don’t need to be pre-defined.

Different websites use different forms of database management systems depending on a range of factors, relational databases such as those constructed with MySQL are a more traditional form of relational database design. MySQL has many key features, the data stored must be in a tabular format and often implement the different normal form structures. MySQL follows a schema design and stores data in different tables joining them when certain queries are required. NoSQL is considered a more modern approach to database management and is used by many large organisations, here the data is not relational and does not confine to the same normal forms and structure as a relational model does. NoSQL doesn’t follow a schema and doesn’t have a fixed design meaning that we have absolute flexibility to add and delete records as and when they are created. NoSQL adopts a denormalization approach where the same data can be stored more than once, this means that data that is read together should be stored together. There is often an important argument regarding what type of database to use for a specific website, these decisions depend upon a wealth of factors and what exactly you need from your database, the use cases for a website are therefore essential to analyse as we can make decisions on what kind of database to implement depending on what use cases are needed.

In this section we will look at the differences between a MySQL and NoSQL design and reference how the Twitter platform would fit into each database structure. Scalability is something that is essential in large databases, as we know Twitter has huge amounts of traffic and has a reported 6,000 tweets per second (2) flowing through their site, therefore there is a lot of data that needs to be stored. NoSQL databases are designed with scalability in mind and can easily be scaled out across many different machines which adhere to the modern cluster computing methods. With Twitter having such huge amounts of data, being able to partition the data out in a sharding process is important, we can easily add nodes to deal with such traffic, this is an important feature and one that comes at a significantly lower cost than scaling up. MySQL, however, is more inclined to be designed to scale up, this means purchasing larger and more expensive machinery to deal with scalability issues, for a company with such huge amounts of data this will come with an extremely high financial cost. Twitter would most definitely benefit from scaling out from a financial perspective.

Scalability is not just a financial issue; it also satisfies many other problems that may arise. External issues such as server failures and destruction of data storage facilities would cause huge negative implications in a scale up structure as having one machine go down would make the whole of Twitter unavailable for a period. However, if these issues arise in a scale out structure there are processes to mitigate these problems, having data stored across many different nodes in different locations means that the risk of failure is spread, replication is also possible in cluster computing so if a group of nodes are unavailable that same data is however available on other nodes so that the data is always made available. As hinted previously in a scale out structure we can store data in different geographic regions with ease, not only does this satisfy different data protection laws it again helps spread the risk of failure, for example if there is a war in a nation and the database storage is destroyed individuals can still access that data through different nodes stored around the globe, this making the distribution of nodes great for redundancy. With NoSQL we can also write to the primary and secondary database servers, since Twitter is heavily read based we can write strictly to the primary server and replication can go to the secondary server which we can read from, this makes the most out distributing reads and writes across the server to achieve maximum efficiency, it also acts favourably for scalability as if the primary server is unavailable for some reason wites can be diverted to the secondary server to keep Twitter running smoothly.

As we have learnt previously Twitter interacts with a huge amount of data constantly, we also know that Twitter needs the latest data for their trending tab, this falls perfectly into the hands of NoSQL database design. Speed and flexibility are therefore two factors that must be adhered too. With MySQL when we need to query data, in many scenarios we need to join the data, joining tables in a MySQL model is without doubt a more timely process than a NoSQL model, this is because a simple use case such as viewing a tweet would require several pieces of information to be joined such as details of the user, contents of the tweet, time, location, likes, retweets etc, these would all fall into different tables under the normalisation framework, hence joining these all together at the rate Twitter operates at would surely not be time effective. Twitter is also a global organisation so having relational databases joining data from different geographic regions around the globe would only further the argument of poor time efficiency. NoSQL can store all this data together in one single object, data that is stored together can be accessed together which vastly reduces the time needed to query said data, this means that data does not need to be joined from many different tables spread across different locations and therefore vastly sharpens the response time of queries.

The use case of writing and posting a tweet with the different databases would be vastly different, MySQL would be effective if the data stored by Twitter was already very structured and there isn’t going to be too much change, the benefits of this is that it would be a rarity to run into errors as the structure is very simple and the data types are pre-defined in a tabular format. Furthermore, the data will be easy to query as it all follows the same format, thus meaning that we can use the same query framework repeatedly, this can help with certain processes where we can design complex functions to be re-used on a regular basis. Twitter however doesn’t have this kind of simple structure, flexibility is needed to store such data as every tweet is different and can be made up of text, numbers, images, videos, etc, as well as needing all the attached information of user interactions. NoSQL lets you add records as and when needed due to its schemeless design, therefore it easily lets you store all these documents together and removes any need for joins. The use case of writing and posting a tweet however can involve a certain level of normalisation in NoSQL, this is achieved by using id’s instead of primary and foreign keys, we may choose to use this as it helps us split different parts of data which are better suited to be stored in their own groups for simplicity and efficiency reasons. The use case identified above may choose to use some level of normalisation such as referencing users who interact with your tweet, from another table which stores further information on each user. It must be noted that we can embed whole documents and link documents through NoSQL, it is also possible to embed partial documents and link the rest of the data in another location where the data can be found, in this situation we could view which user has interacted with our tweet, but we can link extra detail on that specific user in another location.

We must also acknowledge potential loss of data, the use cases of posting or sharing a tweet is important however if the tweet doesn’t post or get shared correctly the consequence are not exactly huge. Inconsistency can arise in a cluster computing environment however such inconsistencies are not detrimental to Twitter, one use case such as updating financial information is extremely important in terms of maintaining its integrity, in this case Twitter must have strict consistency across replicas so that no financial information is lost. With a database design we can used ACID transactions to ensure we have data integrity, however with NoSQL we have the option to strike a balance between consistency and speed depending on what best suits the use case.

When posting tweets, we don’t need this strict consistency, we could settle for casual consistency or even eventual consistency as receiving a tweet straight away isn’t exactly essential, as suggested before we can afford to read from secondary nodes. Twitters trending tab definitely prides itself on the most up to date breaking news without threat of having stale data, therefore the data must be available in a short time period, In this case the Primary/ secondary approach should be implemented so all the writes are sent to the primary node, this not only guarantees consistency because it is important that breaking news is not lost, it also ensures that the data is supplied in a speedy manner. For the use case of receiving tweets we could simply accept that inconsistency across replicas are going to happen, on Twitter some users can see a post before others and that may not be seen as an issue however users of Twitter may become irritated with some use cases such as waiting for a certain brand to post a release of a new product and the tweet is simply not available as inconsistency doesn’t ensure that Tweet even reaches the end user.

With twitter we could store all the data that common queries require together in the same place, the process of storing units of data together is known as aggregate modelling, here we can access common queries through one node due to effective sharding decisions, this process could be used with a use case such as logging into your account, here we can store the username and password together. We also associate Twitter with the freshest data and the top of our feed shows this, with sharding we can store tweets based on creation time, this means that we can store all the most recent tweets together and therefore fetch the top tweets quickly, older tweets may be stored on a disk-based index as tweets from several years ago don’t need to take up space in the RAM as they are not interacted with. The use case of accessing Twitters trending tab to consume the latest information must adhere to the best sharding methods so that we have the most up to date data, we can use an index to speed up the search of data, in which we can use methods such as B-trees so that each entry per node is sorted, this works in a similar way to chapters and page numbers in a book, meaning that we can quickly access the page that we require through providing an indication to the location of the data, this would satisfy any use case that requires data quickly.

Twitter can be used as a direct messaging service, one use case here is simply typing out a message and sending it, with a messaging service it is essential to guarantee security of information, it is also essential to guarantee speed as a conversation between individuals cant flow on slow response times. There must be a balance between speed and security in this case, with a relational database we would require a many to many relationships in a conversation as we can send many messages and receive many messages, having a joining table would cause the process of messaging to be slow, this would be particularly evident if individuals live in a completely different geographic area as the data must join over a longer distance. With NoSQL we can use MongoDB, and in which write to it using JSON, JSON allows us to store many to many relationships in just a single document, this removes the need for normalisation and therefore the need for complex joins. This addresses the issue of speed, with NoSQL the flexibility of having nodes in different locations can ensure that data is summoned with speed.

From the evidence we have analysed so far it is clear that Twitter should implement a NoSQL model rather than a relational design. The main reasons for this are that we have great scalability and speed, these two factors are essential to Twitters platform due to the size of the user base and the need to have up to date data in many situations. Relational models will only limit Twitter as there isn’t much opportunity for great flexibility and to operate on a huge global level.

2.

When designing a NoSQL model, it is important to find a balance between the document size and avoiding the need for joins. Keeping a denormalised approach is still important as certain data should be stored and accessed together, however we can use references to allow for a reduction of document size and for arranging the data so common use case can be best accessed. Reaching such a balance means that we will need to implement some level of normalisation, instead of using joins we reference other collections by their id’s, we are able to embed documents and create links to other documents in this process. Twitter needs to have an effective database design so that it can run effectively, in this section we will create a UML design to represent a document model that Twitter should adhere to. MongoDB is a type of NoSQL document database that uses JSON to store data, that is what we will be using in this section. We must note that JSON stores values in key value pairs, this means we have repetition, but increased flexibility as different entries can have different attributes.

When making decisions regarding a document database design it is important to decide on whether we should embed or reference documents, this is dependant largely on the use cases and the positives or drawbacks of implementing embedding or referencing in relation to those use cases. We can also embed A screenshot of a computer

Description automatically generated with low confidencepartial documents and reference them to other documents to fit certain use cases.

Above we have a database design using a NoSQL model showing the core features of Twitter, one thing to note is that there are several occasions where referencing is used instead of embedding, with a site such as Twitter a post may have tens of thousands of interactions, the larger a document is the more RAM it uses, MongoDB does have a size limit which will often be exceeded on a site with a user base the size of Twitter, hence why embedding was not always an option. This form of database design can be used by front end developers to construct the site, hence certain decisions have been made from a user interface perspective, such as displaying a few pieces of data and referencing to another document where the rest of the data can be found.

The user collection references the followers and following collections, this decision is made as individuals can have millions of followers and separating the data across collections helps to store that, this data can also change frequently where new followers add or delete an individual, this means that writes must be fast to make, which is a great characteristic of a reference. In relation to the tweets collection a document is partially embedded within the tweet framework, this displays a limited amount of information of the interactions embedded document, which would be displayed with the tweets data, the purpose of this is that we have a brief overview and then a reference to where we can find the rest of the data, this decision has been made from a storage perspective and one that breaks down the comments likes and retweets and links them to the user who has interacted with any of those options.

The trending tab uses a different approach, here we embed a document which acts as a preview with reference to additional information. Embedding is often slow to write as it involves updating multiple documents, with the trending tab there is only a limited amount of data to update so speed concern isn’t an issue, we do however benefit from being able to read fast as we don’t have to join and can read from just one document, this is essential for the trending tab as we always want the freshest data. Finally, the direct messaging collection uses references like how the ‘following’ collection works, with referencing there is sometimes an issue of updating and deleting where one collection can be updated but the reference on the other collection does not, for posting a tweet this isn’t so much of a concern as we can easily write to the server to replace any inconsistencies.

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidenceHere we have an example of our user document, this stores information including the id, set to 1, the username, userpassword and useremail. This document was used in reference to others to determine followers and following records. Below is an example of referencing this document and using an element of JavaScript so that the documents are matched on their id’s.

A screenshot of a computer screen

Description automatically generated with medium confidence

Above we have another example of one of the documents, this time we have information regarding an individual’s tweet, this includes supporting information such as location, date, and time.

Twitter should store their data using modern cluster computing techniques, this would involve sharding the data across many different nodes in a scale out approach. We can use indexes to store said data, an index stores memory the way a book has chapters, this could be used in our follower’s collection where we can index based upon alphabetic characters so we can easily locate a follower, MongoDB by default builds an index around the ID field of each collection we however can decide where we want to implement an index. We can also separate indexes in order of N, here we can rule out parts of a search by filtering through the index lengths, here we can search for follower such as splitting through the length-based categories by discarding chunks of entries that don’t fit our specification. We can make use of this idea by implementing a B-Tree, this is the process of splitting data into smaller sections across many nodes, these nodes store data in a sorted manner and ensure that they all store a small amount of entries per node. Indexes stored in RAM can’t grow any bigger than available memory, indexes also have a slow write speed, frequently searched fields must execute quickly when queried which is a concern when implementing indexes. MongoDB lets you build indexes that store the most recent entries in the RAM, and older entries can be put on a disk-based index this should be implemented on Twitter as old tweets from years ago don’t have much purpose other than taking up memory, therefore should be stored in external storage.

A compound index for username and password could be used to store user information, this will let us know if the password checks out as both entries are needed for the index to query. We could also use geospatial indexing in terms of location, this means that we can index based upon where the data storage is so that we have faster query executions in each geographic area. Furthermore, text indexing is a great tool for searching through tweets, this process extracts each word in a field and creates a separate index for each entry, this allows us to search a long text entry using just single words, this would be useful in a Twitter search bar as we can receive data in almost a predictive manner when typing for something on the lines of what we are looking for. We can of course implement sharding decisions so that common queries can be stored and accessed together, shard keys are used to determine where the data is spread, we can store geographically so that data is stored in datacentres closer to where the access points are, this is very much like the geospatial index. It is also possible to split the data using range based sharding this will store the most recent data together and could be useful for storing the most recent tweets. Hash based indexing would however improve on this as it spreads the data across the shards, so each node has the same amount of work to do, this will help balance the huge workload Twitter’s databases have to do. The purpose of indexing is to speed up the process of querying, implementing any of the above processes depending on the use cases identified would be an effective decision for Twitter to take.

References

1. Twitter (2022) Available at: <https://twitter.com/home?lang=en>
2. Anon (2020) Twitter: number of users worldwide 2020, Statista, Available at:

<https://www.statista.com/statistics/303681/twitter-users-worldwide/#:~:text=Twitter%3A%20number%20of%20worldwide%20users%202019%2D2024&text=In%202019%2C%20Twitter's%20audience%20counted,and%20a%20popular%20marketing%20channel>.

1. Sayce, D (2020) The number of tweets per day in 2020, Available at:

<https://www.dsayce.com/social-media/tweets-day/#:~:text=Every%20second%2C%20on%20average%2C%20around%206%2C000%20tweets%20are%20tweeted%20on,200%20billion%20tweets%20per%20year>