Introducing NoSQL and MongoDB

- **Good applications require the capability to store and retrieve data with accuracy, speed, and reliability. Therefore, the data storage mechanism we choose must be capable of performing at a level that satisfies your application's demand.**

- **The three most common data storage solutions available are Direct File System Storage in files, Relational Databases, and NoSQL databases.**

- **The NoSQL data store MongoDB is the most widely used and the most versatile.**

What Is NoSQL?

1. ***NoSQL*** **stands for "Not only SQL," to emphasize the fact that NoSQL databases are an alternative to SQL and can apply SQL-like query concepts.**
2. **NoSQL covers any database that is not a traditional RDBMS.**
3. ***The motivation behind NoSQL*** **is simplified design, horizontal scaling, and finer control over data availability.**
4. **NoSQL databases are more specialized for *types of data*, which makes them more efficient and better performing than RDBMS in most instances.**
5. **NoSQL enable developers to implement models in ways that more closely fit the data flow needs of their system than the traditional relational databases.**
6. ***MongoDB and the Document Model*** **are popular NoSQL technologies because of their great flexibility and scalability in implementing backend storage for web applications and services. MongoDB is a well-supported NoSQL language that is currently available.**

NOSQL DATABASE TYPES

Document Store Databases – In Document store databases, all the data for a single entity can be stored as a document, and documents can be stored together in collections.

- A document can contain all the necessary information to describe an entity. This includes the capability to have subdocuments.
- Documents in the collection are accessed via a unique key.

Key-Value Databases - Key-Value Databases store data in a completely schema-less way, meaning that no defined structure governs what is being stored.

- A key can point to any type of data (from an object, to a string value / to a programming language function.
- Advantage : They are easy to implement and add data to. So, they are efficient to be implemented as simple storage for storing and retrieving data based on a key.

Column Store Databases -Column store databases store data in columns within a key space.

- The key space is based on a unique name, value, and timestamp.
- These databases are geared toward data that uses a timestamp to differentiate valid from stale content.
- This provides the advantage of applying aging to the data stored in the database.

Graph Store Databases - Graph store databases are designed for data that can be easily represented as a graph.

- Elements are interconnected with an undetermined number of relations between them.
  Example: family and social relations, airline route topology.

Choosing RDBMS, NoSQL, or Both

- We might need to implement a strategy based on only RDBMS or NoSQL—or we may find that a combination of the two offers the best solution .
- **With all high-performance databases, we try to balance speed, accuracy and reliability.**

The following is a list of some considerations when choosing a database:

1. **What does my data look like?**
   - Data might favour a table/row structure of RDBMS, a document structure, or a simple key-value pair structure.
2. **How is the current data stored?**
   - If the data is stored in an RDBMS database, we must evaluate what it would take to migrate all or part to NoSQL. Also we may consider whether it is possible to keep the existing data as it is and move forward with new data in a NoSQL database.
3. **How important is the guaranteed accuracy of database transactions?**
   - A disadvantage of NoSQL is that most solutions are not as strong in ACID (Atomic, Consistency, Isolation, Durability) as in the RDBMS systems.

Choosing RDBMS, NoSQL, or Both

4. **How important is the speed of the database?**
   - If speed is the most critical factor for our database, NoSQL might fit our data well and can provide a huge performance boost.
5. **What happens when the data is not available?**
   - Many NoSQL solutions, including MongoDB, provide a good high availability plan using replication and ***sharding*** (Horizontal partitioning of data in database. Each partition stored in different DB server to spread load.)
6. **How is the database being used?**
   - Consider whether most operations on the database are writes to store data or whether they are reads. By this we can define the boundaries of how to split up data, enabling us to gear some data toward writes and other data toward reads.
7. **Should I split up the data to leverage the advantages of both RDBMS and NoSQL?**
   - After the above consideration we might consider putting some of the data, such as critical transactions, in an RDBMS while putting other data, such as blog posts, in a NoSQL database.

Understanding MongoDB

- **MongoDB is an swift and scalable NoSQL database.**
- **MongoDB is based on the NoSQL document store model, in which data objects are stored as separate documents inside a collection.**
- **The documents are stored as binary JSON** (JavaScript Object Notation) **or BSON (**binary encoding of JSON-like documents that MongoDB uses when storing documents in collections. It adds support for data types like Date and binary that aren't supported in JSON.) **objects.**
- ***The Motivation of the MongoDB language*** **is to implement a data store that provides high performance, high availability, and automatic scaling.**
- **MongoDB is extremely simple to install and implement.**
- **MongoDB offers great website back-end storage for high-traffic websites that need to store data such as user comments, blogs, or other items because it is fast, scalable, and easy to implement.**

Why MongoDB has become the most popular NoSQL database:

1. **Document oriented: Because MongoDB is document oriented, the data is stored in the database in a format that is very close to what we will be dealing with in both server-side and client-side scripts. This eliminates the need to transfer data from rows to objects and back.**
2. **High performance: MongoDB is one of the highest-performing databases available. It can support multiple users who interact with the application (websites), having a back end that can support heavy traffic.**
3. **High availability: MongoDB's replication model makes it easy to maintain scalability while keeping high performance and availability.**
4. **High scalability: MongoDB's structure makes it easy to scale horizontally by Sharding the data across multiple servers.**

   In DBMS, Sharding is a type of DataBase partitioning in which a large DataBase is divided or partitioned into smaller data and different nodes. These shards are not only smaller, but also faster and hence easily manageable.
5. **No SQL injection: MongoDB is not vulnerable to SQL injection (putting SQL statements in web forms or other input from the browser that compromises the DB security) because objects are stored as objects, not by using SQL strings.**

Understanding MongoDB - Understanding Collections

- **MongoDB groups data through collections.**
- **A COLLECTION IS SIMPLY A GROUPING OF DOCUMENTS THAT HAVE THE SAME OR A SIMILAR PURPOSE.**
  - **A collection acts similarly to a table in a traditional SQLdatabase.**
- **Difference between MongoDB and Traditional SQL database:**
  - **In MongoDB, a collection is not enforced by a strict schema. Instead, documents in a collection can have a slightly different structure from one another, as needed.**
    - **This reduces the need to break items in a document into several different tables, as in SQL implementations.**

Understanding MongoDB - Understanding Documents

- A document is a representation of a single entity of data in the MongoDB database.
- A collection consists of one or more related objects.
- Records in MongoDB that represent documents are stored as BSON, a lightweight binary form of JSON.
  - It uses field : value pairs that correspond to JavaScript property : value pairs that define the values stored in the document.
- Little translation is necessary to convert MongoDB records back into JSON strings that wemight be using in our application.
- The maximum size of a document in MongoDB is 16MB, to prevent queries that result in an excessive amount of RAM or intensive hits to the file system.
  - You might never come close to this, but you still need to keep the maximum document size in mind when designing some complexdata types that contain file data into your system.

<u>Difference between MongoDB and Traditional SQL database</u>:

- Documents are different from rows. Row data is flat, with one column for each value in the row. However, in MongoDB, documents can contain embedded subdocuments, providing a much closer inherent data model to your applications.

Understanding MongoDB - Understanding Documents

- ***Restrictions in naming Fields***

1. **The field names cannot contain null characters, dots (.), or dollar signs ($).**
2. **The id field name is reserved for the Object ID.**
3. ***The id field is a unique ID for the system that consists of the following parts:***
    a) **A 4-byte value representing the seconds since the last epoch**
    b) **A 3-byte machine identifier**
    c) **A 2-byte process ID**
    d) **A 3-byte counter, starting with a random value**

MongoDB Data Types

- **The BSON data format provides several different types used when storing the JavaScript objects to binary form. These types match the JavaScript type as closely as possible. It is important to understand these types because**
  - **We can query MongoDB to find objects that have a specific property with a value of a certain type.**

    ***Example*: we can look for documents in a database whose timestamp value is a String object or query for ones whose timestamp is a Date object.**
- **MongoDB assigns each data type of an integer ID number from 1 to 255 when querying by type, to identify them.**

TABLE 1.1 MongoDB Data Types and Corresponding ID Number

Type	Number
Double	1
String	2
Object	3
Array	4
Binary data	5
Object ID	7
Boolean	8
Date	9
Null	10
Regular expression	11
JavaScript	13
Symbol	14
JavaScript (with scope)	15
32-bit integer	16
Timestamp	17
64-bit integer	18
Min key	255
Max key	127

- **The order in which different data types in MongoDB are compared when querying to find and update data (When comparing values of different BSON types, MongoDB uses the following comparison order - from lowest to highest) :**

1. Min key (internal type)
2. Null
3. Numbers (32-bit integer, 64-bit integer, double)
4. Symbol, String
5. Object
6. Array
7. Binary data
8. Object ID
9. Boolean
10. Date, timestamp
11. Regular expression
12. Max key (internal type)

Planning Your Data Model

- Before we begin implementing a MongoDB database, we need to understand the nature of the data being stored, how that data will be stored, and how it will be accessed.
    - Understanding these concepts helps us to make determinations ahead of time and structure the data and our application for optimal performance.

Specifically, we should ask ourself the following questions:

1. What basic objects will my application be using?
2. What is the relationship between the different object types—1to 1, 1to many, or many to many?
3. How often will new objects be added to the database?
4. How often will objects be deleted from the database?
5. How often will objects be changed?
6. How often will objects be accessed?
7. How will objects be accessed—by ID, property values, comparisons, or other?
8. How will groups of object types be accessed—common ID, common property value, or other?

When we have the answers to these questions, we are ready to consider the structure of collections and documents inside MongoDB.

Normalizing Data with Document References

- **DATA NORMALIZATION is the process of organizing documents and collections to minimize redundancy and dependency.**
  - **This is done by identifying object properties that are sub-objects and that should be stored as a separate document in another collection from the object's document.**
    - **Typically, this is useful for objects that have a one-to-many or many-to-many relationship with sub-objects.**
- **The Advantage Of Normalizing Data is that the database size will be smaller because only a single copy of objects will exist in their own collection instead of being duplicated on multiple objects in single collection.**
  - **Additionally, if you modify the information in the sub-object frequently, then you need to modify only a single instance instead of every record in the object's collection that has that subobject.**
- **A major Disadvantage Of Normalizing Data is that, when looking up user objects that require the normalized sub-object, a separate lookup must occur to link the sub-object. This can result in a significant performance hit if you are accessing the user data frequently.**

NOTE :$lookup performs an equality match on the localField to the foreignField from the documents of the from collection. If an input document does not contain the localField , the $lookup treats the field as having a value of null for matching purposes

Denormalizing Data with Embedded Documents

- **Denormalizing data is the process of identifying sub-objects of a main object that should be embedded directly into the document of the main object.**
  - **Typically, this is done on objects that have mostly one-to-one relationships or that are relatively small and do not get updated frequently.**
- **The Major Advantage of de-normalized documents is that we can get the full object back in a single lookup without needing to do additional lookups to combine subobjects from other collections. (performance enhancement)**
- **The Disadvantage is that, for sub-objects with a one-to-many relationship, we are storing a separate copy in each document; this slows insertion a bit and takes up additional disk space.**

Using Capped Collections

- A capped collection is a collection that has a fixed size.
- When a new document needs to be written to a collection that exceeds the size of the collection, the oldest document in the collection is deleted and the new document is inserted.
- Capped collections work great for objects that have a high rate of insertion, retrieval, and deletion.

Benefits of using capped collections:

- Capped collections guarantee that the insert order is preserved.
- Queries do not need to use an index to return documents in the order they were stored, eliminating the indexing overhead.
- Capped collections guarantee that the insertion order is identical to the order on disk by prohibiting updates that increase the document size. This eliminates the overhead of relocating and managing the new location of documents.
- Capped collections automatically remove the oldest documents in the collection. Therefore, you do not need to implement deletion in your application code.
- A great use of capped collections is as a rolling log of transactions in your system. You can always access the last *X* number of log entries without needing to explicitly clean up the oldest.

Capped collections impose the following restrictions:

- **You cannot update documents to a larger size after they have been inserted into the capped collection.**
  - **You can update documents, but the data must be the same size or smaller.**

- **You cannot delete documents from a capped collection. The data will take up space on disk even if it is not being used.**
  - **You can explicitly drop the capped collection, which effectively deletes all entries, but you also need to re-create it to use it again.**

Understanding Atomic Write Operations

- **Write operations are atomic at the document level in MongoDB.**
  - **Thus, only one process can be updating a single document or a single collection at the same time.**
  - **This means that writing to documents that are denormalized is atomic.**
  - **However, writing to documents that are normalized requires separate write operations to subobjects in other collections; therefore, the write of the normalized object might not be atomic as a whole.**
- **You need to keep atomic writes in mind when designing your documents and collections to ensure that the design fits the needs of the application.**
  - **In other words, if you absolutely must write all parts of an object as a whole in an atomic manner, you need to design the object in a denormalized way.**

Considering Document Growth

When you update a document, you must consider what effect the new data will have on document growth.

- **MongoDB provides some padding in documents to allow for typical growth during an update operation. However, if the update causes the document to grow to a size that exceeds the allocated space on disk, MongoDB must relocate that document to a new location on the disk, incurring a performance hit on the system.**
- **Frequent document relocation also can lead to disk fragmentation issues.**
  - **For example, if a document contains an array and you add enough elements to the array to exceed the space allocated, the object needs to be moved to a new location on disk.**
- **One way to mitigate document growth is to use normalized objects for properties that can grow frequently.**
  - **For example instead of using an array to store items in a Cart object, you could create a collection for CartItems ; then you could store new items that get placed in the cart as new documents in the CartItems collection and reference the user's cart item within them.**

Identifying Indexing, Sharding, and Replication Opportunities

MongoDB provides several mechanisms to optimize performance, scale, and reliability. As you are contemplating your database design, consider the following options:

- **Indexing: Indexes improve performance for frequent queries by building a lookup index that can be easily sorted.**
  - **The id property of a collection is automatically indexed on because looking up items by ID is common practice.**
- **Sharding: Sharding is the process of slicing up large collections of data among multiple MongoDB servers in a cluster.**
  - **Each MongoDB server is considered a shard. This provides the benefit of utilizing multiple servers to support a high number of requests to a large system.**
  - **This approach provides horizontal scaling to your database.**
    - ✓ **You should look at the size of your data and the amount of request that will be accessing it to determine whether to shard your collections and how much to do so.**
- **Replication: Replication is the process of duplicating data on multiple MongoDB instances in a cluster.**
  - **When considering the reliability aspect of your database, you should implement replication to ensure that a backup copy of critical data is always readily available**

Large Collections vs. Large Numbers of Collections

- **Important consideration when designing your MongoDB documents and collections is the number of collections the design will result in.**
  - **Having a large number of collections doesn't result in a significant performance hit, but having many items in the same collection does.**
  - **Consider ways to break up your larger collections into more consumable chunks.**
    - **An example of this is storing a history of user transactions in the database for past purchases.**
      - **You recognize that, for these completed purchases, you will never need to look them up together for multiple users.**
      - **You need them available only for users to look at their own history.**
      - **If you have thousands of users who have a lot of transactions, storing those histories in a separate collection for each user makes sense.**

Deciding on Data Life Cycles

One of the most commonly overlooked aspects of database design is the data life cycle. How long should documents exist in a specific collection?

- **Some collections have documents that should be kept indefinitely (for example, active user accounts).**
- **However, each document in the system incurs a performance hit when querying a collection.**
- **We should define a *Time To Live (TTL)* value for documents in each of your collections.**
- **We can implement a TTL mechanism in MongoDB in several ways.**
  - **One method is to implement code in your application to monitor and clean up old data.**
  - **Another method is to utilize the MongoDB TTL setting on a collection, to define a profile in which documents are automatically deleted after a certain number of seconds or at a specific clock time.**
  - **Another method for keeping collections small when you need only the most recent documents is to implement a capped collection that automatically keeps the size of the collection small.**

Considering Data Usability and Performance

These are the two most important aspects of any web solution and, consequently, the storage behind it.

- **Data usability describes the capability of the database to satisfy the functionality of the website. You need to make certain first that the data can be accessed so that the website functions correctly.**
  - **Users will not tolerate a website that does not do what they want it to. This also includes the accuracy of the data.**
- **Performance: Your database must deliver the data at a reasonable rate.**
  - **We can consult the previous sections when evaluating and designing the performance factors for your database.**
- **In some more complex circumstances, we might find it necessary to evaluate data usability, then consider performance, and then look back to usability in a few cycles until you get the balance correct.**
- **Usability requirements can change at any time. We must be sure to design our documents and collections so that they can become more scalable in the future, if necessary.**