Week-7-TLM

MongoDB Introduction

MongoDB is a document-oriented NoSQL database used for high volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents. Documents consist of key-value pairs which are the basic unit of data in MongoDB. Collections contain sets of documents and function which is the equivalent of relational database tables. MongoDB is a database which came into light around the mid-2000s.

MongoDB allows a highly flexible and scalable document structure. For e.g. one data document in MongoDB can have five columns and the other one in the same collection can have ten columns. Also, MongoDB database are faster as compared to SQL databases due to efficient indexing and storage techniques.

MongoDB Features

Each database contains collections which in turn contains documents. Each document can be different with a varying number of fields. The size and content of each document can be different from each other.

The document structure is more in line with how developers construct their classes and objects in their respective programming languages. Developers will often say that their classes are not rows and columns but have a clear structure with key-value pairs.

The rows (or documents as called in MongoDB) doesn’t need to have a schema defined beforehand. Instead, the fields can be created on the fly.

The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.

Scalability – The MongoDB environments are very scalable. Companies across the world have defined clusters with some of them running 100+ nodes with around millions of documents within the database

MongoDB Example

The below example shows how a document can be modeled in MongoDB.

The _id field is added by MongoDB to uniquely identify the document in the collection.

What you can note is that the Order Data (OrderID, Product, and Quantity) which in RDBMS will normally be stored in a separate table, while in MongoDB it is actually stored as an embedded document in the collection itself. This is one of the key differences in how data is modeled in MongoDB.

Why Use MongoDB?

Below are the few of the reasons as to why one should start using MongoDB

Document-oriented – Since MongoDB is a NoSQL type database, instead of having data in a relational type format, it stores the data in documents. This makes MongoDB very flexible and adaptable to real business world situation and requirements.

Ad hoc queries – MongoDB supports searching by field, range queries, and regular expression searches. Queries can be made to return specific fields within documents.

Indexing – Indexes can be created to improve the performance of searches within MongoDB. Any field in a MongoDB document can be indexed.

Replication – MongoDB can provide high availability with replica sets. A replica set consists of two or more mongo DB instances. Each replica set member may act in the role of the primary or secondary replica at any time. The primary replica is the main server which interacts with the client and performs all the read/write operations. The Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically switches over to the secondary and then it becomes the primary server.

Load balancing – MongoDB uses the concept of sharding to scale horizontally by splitting data across multiple MongoDB instances. MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure.

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document**A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.**

Difference between MongoDB & RDBMS

.RDBMS	**MongoDB**
Database	Database

Table	Collection
Tuple/Row	Document
column	Field

- **Data Modelling in MongoDB**

As we have seen from the Introduction section, the data in MongoDB has a flexible schema. Unlike in SQL databases, where you must have a table's schema declared before inserting data, MongoDB's collections do not enforce document structure. This sort of flexibility is what makes MongoDB so powerful.

- When modeling data in Mongo, keep the following things in mind
- What are the needs of the application – Look at the business needs of the application and see what data and the type of data needed for the application. Based on this, ensure that the structure of the document is decided accordingly.
- What are data retrieval patterns – If you foresee a heavy query usage then consider the use of indexes in your data model to improve the efficiency of queries.
- Are frequent inserts, updates and removals happening in the database? Reconsider the use of indexes or incorporate sharding if required in your data modeling design to improve the efficiency of your overall MongoDB environment.

Difference between MongoDB & RDBMS

- Apart from the terms differences, a few other differences are shown below
- Relational databases are known for enforcing data integrity. This is not an explicit requirement in MongoDB.
- RDBMS requires that data be normalized first so that it can prevent orphan records and duplicates Normalizing data then has the requirement of more tables, which will then result in more table joins, thus requiring more keys and indexes.

- As databases start to grow, performance can start becoming an issue. Again this is not an explicit requirement in MongoDB. MongoDB is flexible and does not need the data to be normalized first.

- **Document Schemas**

- A **document schema** is a JSON object that allows you to define the shape and content of documents and embedded documents in a collection. You can use a schema to require a specific set of fields, configure the content of a field, or to validate changes to a document based on its beginning and ending states.

- Document schemas follow the same JSON schema specification as document validation in the MongoDB server

- **Data Modeling Introduction**

- The key challenge in data modeling is balancing the needs of the application, the performance characteristics of the database engine, and the data retrieval patterns. When designing data models, always consider the application usage of the data (i.e. queries, updates, and processing of the data) as well as the inherent structure of the data itself.

- Flexible Schema

- Unlike SQL databases, where you must determine and declare a table's schema before inserting data, MongoDB's collections, by default, do not require their documents to have the same schema. That is:

- The documents in a single collection do not need to have the same set of fields and the data type for a field can differ across documents within a collection.
- To change the structure of the documents in a collection, such as add new fields, remove existing fields, or change the field values to a new type, update the documents to the new structure.

- This flexibility facilitates the mapping of documents to an entity or an object. Each document can match the data fields of the represented entity, even if the document has substantial variation from other documents in the collection.

- In practice, however, the documents in a collection share a similar structure, and you can enforce document validation rules for a collection during update and insert operations. See Schema Validation for details.