Day - 2

MongoDB internal Architecture

Ajit Yadav (DBA)

In this topic we have discuss the evolution of MongoDB internal architecture on how documents are stored and retrieved focusing onthe index storage representation.
I assume reader is well versed with fundamentals of database engineering such as indexes, B+Trees, data files, WAL etc.
Note : If don't know about this i'll clear in further lectures.

MongoDB Architecture Overview

MongoDB's architecture is built to handle the needs of modern applications by providing flexibility, scalability, and performance

Applications

Applications interacting with MongoDB can be built in various languages like Python, .NET, and Java. These applications typically access data stored on a
MongoDB server. They interact with MongoDB through drivers, which communicate directly with the MongoDB server.

When an application requests data, the MongoDB driver processes the query, connects to the MongoDB server, and executes the query through MongoDB's query engine.

Drivers

MongoDB drivers serve as the interface between your application and the MongoDB database. They act as a bridge, translating application-level operations into
MongoDB commands, allowing developers to interact with the database in a seamless way.

Drivers provide an API (Application Programming Interface) for CRUD operations—Create, Read, Update, and Delete—allowing you to manage data efficiently.

How MongoDB Drivers Work

Initialization and Configuration: Applications initialise MongoDB drivers using a client object, configuring options like timeouts, authentication, and connection settings.

Executing Database Operations: Drivers execute CRUD operations, transactions, and aggregations against the database.

Handling Data Formats: MongoDB drivers convert data into BSON format for storage and back into native formats when retrieved.

Transaction Management: Drivers ensure operations are executed as part of a transaction, with all-or-nothing commit or rollback.

Closing Connections: When applications shut down, drivers manage the proper closure of active connections and clean up resources.

Note: Drivers are part of the application stack and not the MongoDB server. They enable applications to communicate with MongoDB efficiently.

MongoDB Query Engine
The MongoDB Query Engine is responsible for processing, optimizing, and executing queries. It handles both simple and complex queries, making data retrieval quick and efficient.

How the Query Engine Works :

Query Parsing: When a query is received, it is parsed to check its syntax and is converted into an internal format that MongoDB understands.

Query Planning: MongoDB generates multiple query plans based on available indexes, query filters, and sort order. The query planner then selects the most efficient plan.

Query Execution: The selected plan is executed, retrieving the necessary data.

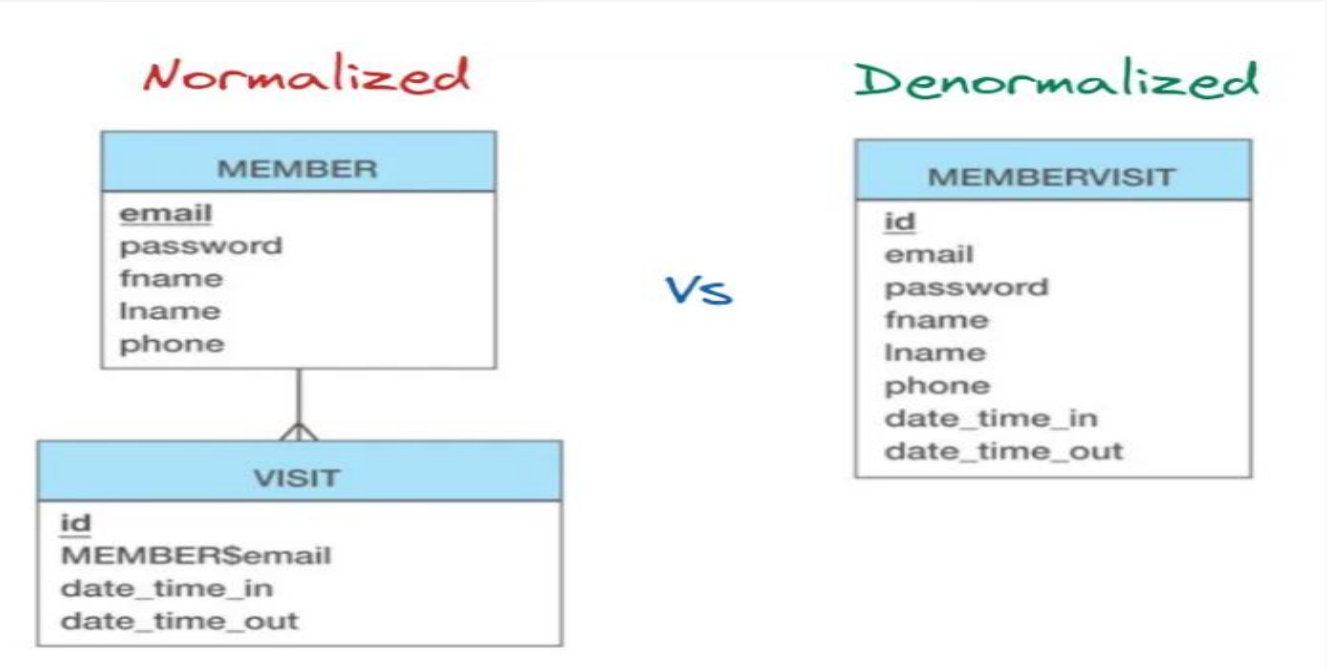
Result Optimization: The engine continuously refines query execution strategies based on performance metrics, ensuring queries run efficiently over time.

Scenario : Consider a query that requests all users over the age of 30. The query engine parses the query, identifies indexes on the age field, selects the best query plan, and retrieves the relevant documents from the database.

MongoDB data model

The MongoDB data model defines how data is stored and relationships are maintained between documents.

MongoDB's flexible schema allows data to be structured in two key ways:



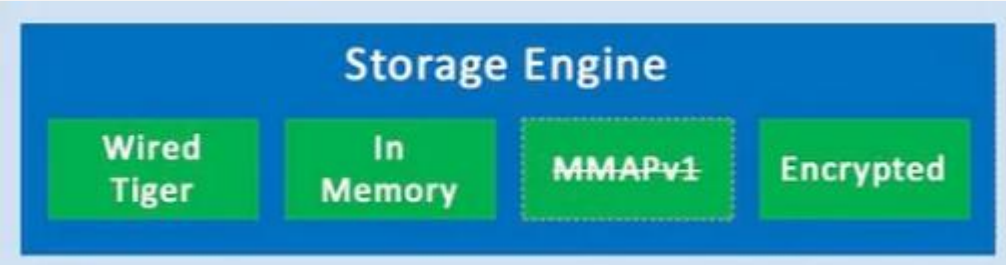
1. Embedded Data Model

Embedded Data Model (also called a nested document) means storing related data within the same document rather than in a separate collection. Instead of using foreign keys like in SQL, MongoDB lets you embed documents inside other documents. This works well for representing "contains" or one-to-few relationships.

2. Normalized Data Model

Normalized Data Model means storing related data in separate collections and linking them using references, similar to how relational databases work with foreign keys.

Storage Engine



MongoDB's storage engine is responsible for managing how data is stored both in memory and on disk. MongoDB supports several storage engines, including:

MMAPv1: The original storage engine, now mostly deprecated.
Wired Tiger: The default storage engine, designed for high-concurrency workloads.
In Memory: Optimized for in-memory storage when low-latency access is critical.

Wired Tiger Storage Engine
WiredTiger became the default storage engine in MongoDB starting from version 3.2 due to its ability to handle high-concurrency workloads efficiently. It provides features like compression, encryption, and document-level concurrency control, which improves performance for modern applications.

Why WiredTiger is the default Storage Engine ?
WiredTiger became the default storage engine for MongoDB due to several advantages it holds over MMAPv1, the original storage engine. WiredTiger storage engine offers document-level concurrency control, whereas MMAPv1 has collection-level locking.

It also has a write-ahead log, making it more resistant to crashes. WiredTiger uses prefix compression, reducing storage costs for larger datasets. Overall, it performs better for random writes and operational databases with low latency and transactional workloads due to its B-tree structure and well-ordered data structure.

WiredTiger Engine
Components

Schema & Cursors:

Schema : Represents the structure of the data and it also defines how data is stored and organized in the database.

Cursors : It allow for sequential iteration through data and facilitate navigating through the stored data efficiently.

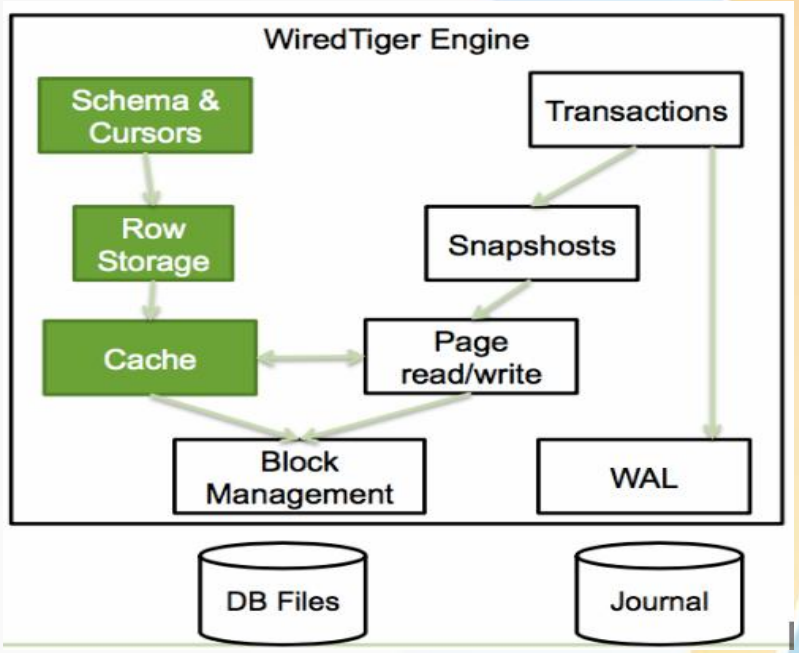
Row storage and Column storage:
WiredTiger supports these two storage.

Row Storage : Data is stored in a row-wise format, where entire rows of a table are stored together.This format is ideal for read and write heavy workloads.
Column Storage : Data is stored column-wise, where each column's data is stored separately.It is often used for
analytical workloads.It also improves the speed of aggregation queries by reducing the amount of data read from disk.

Cache:
It is used to store frequently accessed data and metadata in memory.The cache allows faster reads by reducing the need to access the disk and improving performance. When the cache fills up, it pushes the least-recently-used data to disk.

Block Management:
Handles the allocation and de-allocation of disk space for storing data. Block management ensures that data is stored efficiently on disk and handles tasks such as freeing up space when data is deleted.

Page read / write:
Pages are the smallest units of data read from or written to the disk. WiredTiger reads and writes data in pages rather than individual records to improve efficiency.



Transactions:
It ensures ACID properties for all operations.It allows multiple operations to be grouped together, ensuring that the data remains consistent, even in
case of failure.

Snapshots:
Snapshots create consistent views of the data at a particular point in time,
allowing readers to see a consistent view of the data while writes are being
performed. This helps in ensuring high concurrency by reducing locking
contention.

Logging:
The logging component in WiredTiger is crucial for crash recovery and
durability. It maintains a record of all changes made to the database.

Database files & Log files

These files are mainly used for recovery purpose.

Database files : These files store the actual data (documents or records). WiredTiger manages how the data is organized and stored in these files on disk.

Log files(Journal) : These store the transaction logs, which are used for crash recovery and ensuring durability.Making sure that committed transactions are saved, even in the event of a system failure.