# Introduction

## Abstract

In today’s era, organizations are developing applications, which continuously and rapidly generates large amount of data. In this world of big data, NoSQL databases are rapidly becoming popular for storing information among organizations. Therefore, it is essential for an organization to choose a database which is compatible and efficient for their applications. To choose a correct database, it is essential to examine the performance of various databases under diverse workload conditions.

## General Introduction

Today, applications and information system need to manage a large amount of different data while meeting various requirements such as performance and scalability. NoSQL system provide efficient data management solutions while offering flexibility in structuring data. Our work focuses on document-oriented database systems, specifically those storing JSON documents, including MongoDB. These systems are schema-free. They support semi-structured data without a previous creation of a schema (unlike relational DBMS). Data can be stored in collections of documents with atomic and complex attributes. This flexibility enables rapid initial development and permits many data structure possibilities for the same information. The choice is quite crucial for its potential impact on several aspects of application quality. Indeed, each structure may have advantages and disadvantages regarding several aspects, such as the memory footprint of the document base, data redundancy, navigation cost, data access or program readability and maintainability.

## Overview of Concepts

### 1.2.1 Database

A database is an organized collection of structured information, or data, typically stored electronically in a computer system . Databases are structured to facilitate the storage, retrieval, modification, and deletion of data in conjunction with various data-processing operations.

A database is usually controlled by a database management system (DBMS). Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to just database.

### 1.2.2 NoSQL databases

NoSQL is an approach to database management that can accommodate a wide variety of data models, including key-value, document, columnar and graph formats. A NoSQL database generally means that it is non-relational, distributed, flexible and scalable. Additional common NoSQL database features include the lack of a database schema, data clustering, replication support and eventual consistency, as opposed to the typical ACID (atomicity, consistency, isolation and durability) transaction consistency of relational and SQL databases. Many NoSQL database systems are also open source.

The term NoSQL originally could be taken at its word -- that is, SQL was not used as the API to access data. However, the ubiquity and usefulness of SQL caused many NoSQL databases to add support for SQL. Today it is commonly accepted that NoSQL stands for "Not Only SQL."

There are four popular types of NoSQL database systems. Each uses a different type of data model, resulting in significant differences between each NoSQL type.

Graph databases: Graph data stores organize data as nodes, which are similar to rows in a relational database, and edges, which represent connections between nodes.

Key-value stores: Also known as key-value databases, these systems implement a simple data model that pairs a unique key with an associated value.

Wide-column stores: These databases use familiar tables, columns and rows like relational database tables, but column names and formatting can differ from row to row in a single table.

Document databases: Also called document stores, these databases store semi-structured data and descriptions of that data in document format.

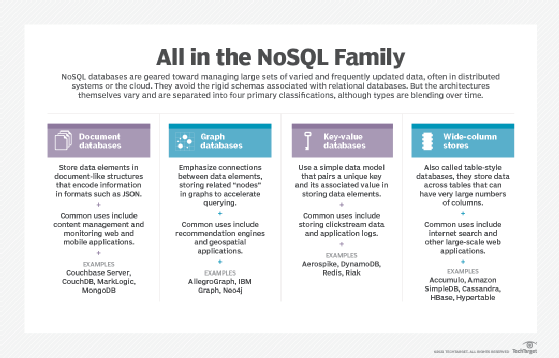


Figure 1: The four Main Types of Document oriented databases

These basic NoSQL database classifications are only guides. Over time, vendors have mixed and matched elements from different NoSQL database families to achieve more generally useful systems. That evolution is seen, for example, in MarkLogic, which added a graph store and other elements to its original document databases. Couchbase Server supports both key-value and document approaches. Cassandra has combined key-value elements with a wide-column store and a graph database. Sometimes NoSQL elements are mixed with SQL elements, creating a variety of databases that are referred to as multi-model databases.

This research is going to cluster around the most used and most popular type of NoSQL database which is the **Document Oriented database.**

### 1.2.3 Document Oriented databases

A document-oriented database is a type of NoSQL database in which data is stored in binary document files. This type of database associates each document with a unique key that takes the form of a string or path. Keys are used to locate and pull individual documents from the database. A document-oriented database may also be referred to as a Document Store.

The way documents are organized in the database will also differ depending on the document content. Typically, documents are organized by tags, metadata or collection. An important advantage of using a document store is that if at some future time the data model needs to be change, only the affected documents will have to be updated.

Document database typically have an API or query language that allows developers to execute the create, read, update and delete operations (CRUD).

Some popular document-oriented database include; MongoDB, DynamoDB, Couchbase, Amazon DocumentDB and Cosmos DB.

**DynamoDB:** It is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. It allows users to create databases capable of storing and retrieving any amount of data and comes in handy while serving any amount of traffic.

**DynamoDB**: It is a fully managed NoSQL database service provided by Amazon Web Services. A user can interact with the service by using AWS Management Console or a DynamoDB API. The service also provides users with a high level of I/O performance. Every DynamoDB is executed by a primary key identifier, which can locate each item.

**Couchbase:** Couchbase is a NoSQL distributed document database with many of the best features of a relational DBMS. It is built for micro services and for serverless, consumption-based computing on the cloud, delivering edge computing for mobile and IoT devices that are connected only occasionally or locally. Couchbase serves the modern needs of application developers by providing the greatest strengths of SQL, NoSQL, and NewSQL in a single database.

**MongoDB**: MongoDB is a cross platform document database with the scalability and flexibility that you want with the querying and indexing that you need. It is one of the most popular examples of a document-oriented database. It includes features such as full index support, replication and sharing. A core function of this DB is its horizontal scalability, which makes it a useful DB for companies running big data applications.

### 1.2.4 Documents

A document is a record in a document database. A document typically stores information about one object and any of its related metadata.

Documents store data in field-value pairs. The values can be a variety of types and structures, including strings, numbers, dates, arrays, or objects. Documents can be stored in formats like JSON, BSON, and XML.

### 1.2.5 Collections

A collection is a group of documents. Collections typically store documents that have similar contents. Not all documents in a collection are required to have the same fields, because document databases have a flexible schema. Note that some document databases provide schema validation, so the schema can optionally be locked down when needed.

# Objectives and Context

## 2.1 Objective of the study

There are two main Objectives in this study which are listed in the following lines

* To investigate how document-oriented databases can be used for specific applications such as a blog website with a popular document oriented database such as MongoDB.
* evaluate the performance of document-oriented databases compared to relational databases in terms of scalability and flexibility. This evaluation will be done using the popular MongoDB as the document oriented database and MySQL as the relational database.

## 2.2 Context

Document-oriented databases are becoming increasingly popular due to their flexibility, scalability, and ease of use. However, there is still much to learn about how these databases can be used effectively in different applications. This research aims to investigate the use of document-oriented databases for blog websites and evaluate their performance in terms of scalability and flexibility. The study will explore the advantages and disadvantages of using document-oriented databases for this purpose and compare them with other types of databases. The results of this research will provide valuable insights into the use of document-oriented databases for blog websites and help developers make informed decisions about which type of database to use for their applications.

# Methodology and Solution

## 3.1 Methodology

To create a solution that will permit us to achieve our objectives, we had to pass through a methodology which specifies the procedure to follow to develop a quality, maintainable and performant solution.

### 3.1.1 RECALL OF OBJECTIVES

The main objective of this research is to investigate how document-oriented databases can be used for specific applications such as a blog website with a popular document oriented database such as MongoDB. It is based on this objective that our methodology was chosen and applied.

## 3.1.2 Modelling techniques

The traditional modeling technique used for designing Relational databases are the:

* Entity relationship diagram (ERD) fro the conceptual design.
* Relational Schema for the logical schema
* The conceptual data model (MCD) in MERISE
* The Logical data model (MLD) in MERISE

These are good techniques for modelling databases but it has a drawback which is the fact that it drives developers to develop their application based on the constraints in the database.

But with NoSQL databases and more precisely document oriented databases, we have a different approach called the document oriented modeling which is advantageous to developers because in this approach, we develop the database based on the application and not the contrary as in relational modelling.

### 3.1.2.1 Document Oriented Modeling

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, collections in a DODB, by default, do not require their document 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

### Document Structure

The key decision in designing data models for MongoDB applications revolves around the structure of documents and how the application represents relationships between data. MongoDB allows related data to be embedded within a single document.

Before deciding the documents structures you need to analyse the type of relationship that exist between the collections. There are 3 types of relationship:

* One to one relationship
* One to many relationship
* Many to many relationship

Relationships between documents can be established either by embedding child documents in parent documents or by referencing.

Effective data models support your application needs. The key consideration for the structure of your documents is the decision to embed or to use references.

### Embedded Data (denormalized data model)

* Embedded documents capture relationships between data by storing related data in a single document structure.
* MongoDB documents make it possible to embed document structures in a field or array within a document.
* These denormalized data models allow applications to retrieve and manipulate related data in a single database operation.



Figure 2: Example of embedded documents

In general, use embedded data models when:

* You have “contains” relationships between entities that Is a one to one relationship.
* You have one-to-many relationships between entities. In these relationships the “many” or child documents always appear with or are viewed in the context of the “one” or parent documents

In general, embedding provides better performance for read operations, as well as the ability to request and retrieve related data in a single database operation. Embedded data models make it possible to update related data in a single atomic write operation.

For many use cases in DODB, the denormalized data model is optimal.

### References (Normalized data models)

References store the relationships between data by including links or references from one document to another. Applications can resolve these references to access the related data. Broadly, these are normalized data models.

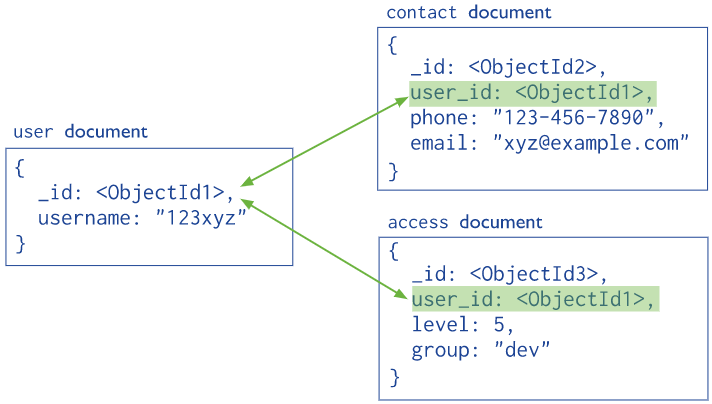


Figure 3: Example of a normalized data model for a DODB

In general, use normalized data models:

* When embedding would result in duplication of data but would not provide sufficient read performance advantages to outweigh the implications of the duplication.
* To represent more complex many-to-many relationships.
* To model large hierarchical data sets.

When developing a data model, analyze all of your application's read and write operations in conjunction with the following considerations.

* **Atomicity:** a write operation is atomic on the level of a single document, even if the operation modifies multiple embedded documents within a single document. When a single write operation modifies multiple documents, the modification of each document is atomic, but the operation as a whole is not atomic.
* **Embedded documents:** The embedded data model combines all related data in a single document instead of normalizing across multiple documents and collections. This data model facilitates atomic operations.
* **Multi-Document Transaction:** For data models that store references between related pieces of data, the application must issue separate read and write operations to retrieve and modify these related pieces of data.