# Non-relational DBMS (NoSQL) Report

**01286241 Database System**

**Software Engineering Program**

**Faculty of Engineering, KMITL**

By

65011277 Chanasorn Howattanakulphong

65011320 Kanokjan Singhsuwan

65011381 Napatr Sapprasert

## Table of Contents

## Introduction

The advent of NoSQL databases has revolutionized the way data is stored, managed, and queried in modern software systems. Traditional relational databases, while effective for structured data, often struggle to accommodate the increasing volume, variety, and velocity of data generated in today's digital landscape. NoSQL databases offer a compelling alternative, embracing diverse data models and dispensing with rigid schemas to meet the evolving needs of modern applications.

Among the array of NoSQL options, MongoDB stands out as a leading document-oriented database management system (DBMS) renowned for its flexibility, scalability, and performance. Unlike relational databases, which organize data into tables with fixed schemas, MongoDB stores data in flexible, JSON-like documents organized into collections. This document-oriented approach enables developers to model complex, heterogeneous data more naturally, facilitating rapid application development and iteration.

MongoDB's appeal extends beyond its data model to encompass a rich ecosystem of features designed to enhance developer productivity and operational efficiency. From schema validation to comprehensive indexing and aggregation capabilities, MongoDB offers a robust toolkit for building high-performance, resilient applications capable of handling diverse workloads and scaling effortlessly to meet growing demand.

In this report, we delve into MongoDB's key features, focusing on its logical data structure, integrity constraints, and querying capabilities. By examining these facets in conjunction with examples drawn from a hypothetical "Presidential Database," we aim to elucidate MongoDB's practical utility and illustrate its applicability across a range of use cases.

## **1. Describe its logical data structure(s), integrity constraints, and data manipulation and data definition languages.**

### Data structure :

Instead of using traditional rows and columns like in a spreadsheet, MongoDB stores the data in a format similar to JSON (JavaScript Object Notation) called "document".

These documents are organized into collections that contain similar types of information. Each collection can hold multiple documents, which are like individual records or entries. Collections can be queried using MongoDB Query Language (MQL).

In relational databases, you usually have to define the structure of your data (the schema), defining the attributes first before you can start storing information. However, in this case, it's not mandatory. This means you can start storing data without specifying exactly what each document will look like in advance. It provides flexibility because you can add new fields or change the structure as needed without strict rules upfront.

## Integrity constraints:

In MongoDB, integrity constraints are enforced through a feature called "Schema Validation". While MongoDB is often referred to as a "schema-less" database because it allows flexibility in document structures within a collection, schema validation allows you to define rules to enforce data integrity on a per-collection basis.

Here's an overview of how integrity constraints are managed in MongoDB using schema validation:

### Schema Validation in MongoDB:

Document Structure Validation: You can specify the structure that documents in a collection must adhere to. This includes defining required fields, specifying data types, and setting constraints on the values allowed for certain fields.

Validation Rules: MongoDB supports a variety of validation rules that can be applied to fields within documents. These rules include options for enforcing data types, defining custom validation logic using JavaScript expressions, and validating the presence or absence of fields.

Error Handling: If a document fails validation against the defined schema rules, MongoDB can be configured to reject the write operation, preventing the insertion or modification of invalid data.

Example of Schema Validation:

JavaScript
```javascript
db.createCollection("employees", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["name", "age", "department"],
      properties: {
        name: {
          bsonType: "string",
          description: "must be a string and is required"
        },
        age: {
          bsonType: "int",
          description: "must be an integer and is required"
        },
        department: {
          bsonType: "string",
          description: "must be a string and is required"
        }
      }
    }
  }
})
```

In this example, a collection named "employees" is created with schema validation. It specifies that each document must have attributes "name", "age", and "department" fields, and each of these fields must adhere to specific data types.

Data manipulation and data definition languages

In MongoDB, data manipulation is primarily done through CRUD operations, which stands for Create, Read, Update, and Delete. These operations allow you to interact with the data stored in MongoDB collections. The data manipulation language (DML) used for querying and modifying data is called MQL, and the data definition language (DDL) is used for defining and managing the structure of the database and its collections.

## Data Manipulation Language (DML):

MongoDB uses a query language similar to JSON called the MongoDB Query Language (MQL). Some common operations include:

Insert: To insert data into a collection, use the insertOne() or insertMany() methods are used.

```javascript
db.users.insertOne({
  name: "Alice",
  age: 30,
  email: "alice@example.com"
});



db.users.insertMany([
  { name: "Alice", age: 30, email: "alice@example.com" },
  { name: "Bob", age: 35, email: "bob@example.com" },
  { name: "Charlie", age: 25, email: "charlie@example.com" }
]);
```

Find: To query data from a collection, the find() method is used. It allows the specification of query criteria and projection fields.

```JavaScript
db.users.find({ age: { $gt: 25 } }, { name: 1, age: 1 });
```

Update: To update existing documents in a collection, updateOne() or updateMany() methods are used.

```JavaScript
db.users.updateOne(
  { name: "Alice" },
  { $set: { age: 31 } }
);
```

```
// Increase the age of all users by 1
db.users.updateMany({}, { $inc: { age: 1 } });
```

Delete: To delete documents from a collection, deleteOne() or deleteMany() methods are used.

```JavaScript
db.users.deleteOne({ name: "Alice" });


// Delete all documents where age is less than 30
db.users.deleteMany({ age: { $lt: 30 } });
```

**Data Definition Language (DDL):**

In MongoDB, DDL operations are primarily focused on defining and managing collections. MongoDB is schema-less, meaning there is no need to explicitly define the structure of a collection beforehand. However, it allows for operations to create and manage indexes, which are crucial for optimizing query performance.

Create Collection: it allows for creating a collection explicitly using the createCollection() method. By default, collections are created automatically when you insert data into them.

```JavaScript
db.createCollection("products");
```

Drop Collection: To drop a collection and delete all of its documents, drop() method is used.

```JavaScript
db.products.drop();
```

Create Index: Indexes can be created to improve the performance of queries. using the createIndex() method is used to create indexes on one or more fields.

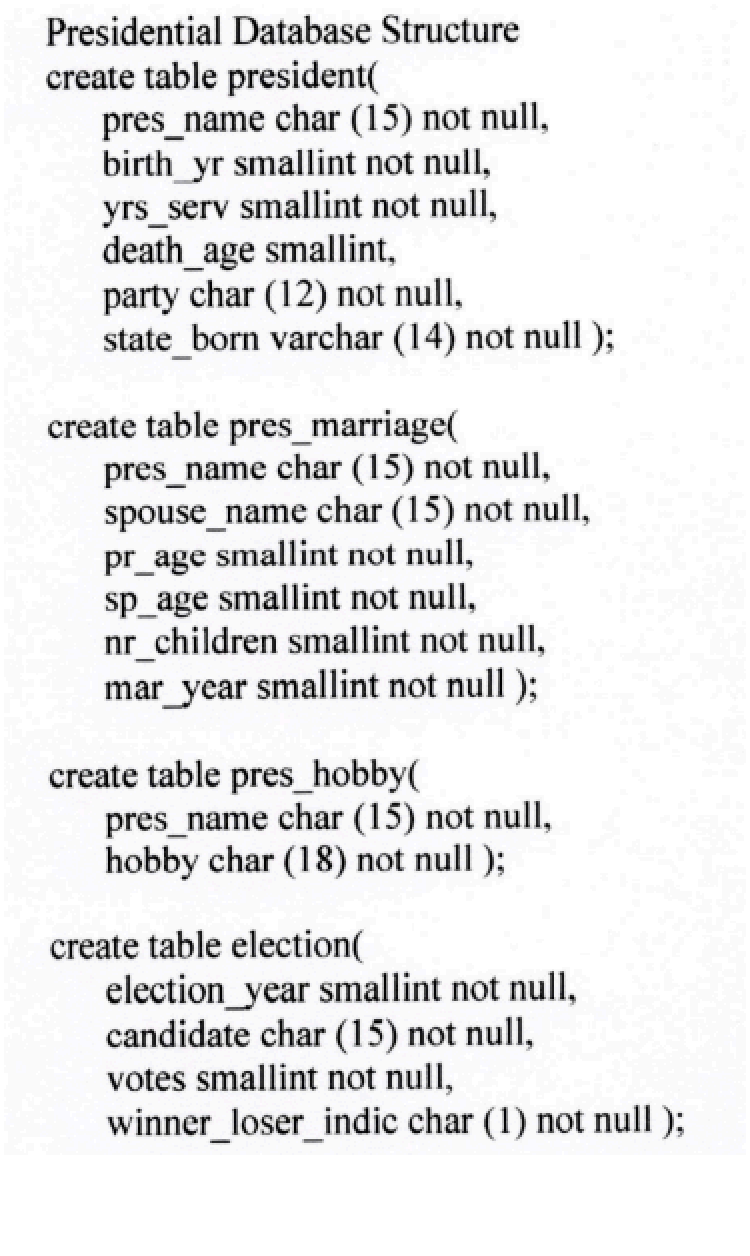
```JavaScript
db.products.createIndex({ name: 1 });
```

Drop Index: To drop an index, the dropIndex() method is used.

```JavaScript
db.products.dropIndex("name_1");
```

**2. Demonstrate some queries using its NoSQL query language on the Presidential Database (the database used in our SQL lectures). Compare the querying ability of the language to SQL.**

MongoDB query language Examples



Presidential Database Structure

```
create table president(
     pres_name char (15) not null,
     birth_yr smallint not null,
     yrs_serv smallint not null,
     death_age smallint,
     party char (12) not null,
     state_born varchar (14) not null );

create table pres_marriage(
     pres_name char (15) not null,
     spouse_name char (15) not null,
     pr_age smallint not null,
     sp_age smallint not null,
     nr_children smallint not null,
     mar_year smallint not null );

create table pres_hobby(
     pres_name char (15) not null,
     hobby char (18) not null );

create table election(
     election_year smallint not null,
     candidate char (15) not null,
     votes smallint not null,
     winner_loser_indic char (1) not null );
```

Example of MongoDB query MQL:

1. Find presidents who served between 1800 and 1900:

```JavaScript
db.president.find({
 yr_serv: {
   $gte: 1800, //greater or equal
   $lte: 1900 //lesser or equal
 }
})
```

Equivalent SQL code:

```Unset
SELECT *
FROM president
WHERE yr_serv BETWEEN 1800 AND 1900;
```

2. Find the president with the highest age.

JavaScript
```
db.president.aggregate([
 {
   $project: {
     pres_name: 1,
     age: { $subtract: ["$death_age", "$birth_yr"] }
   }
 },
 {
   $sort: { age: -1 }
 },
 {
   $limit: 1
 }
])
```

Equivalent SQL code:

Unset
```
SELECT pres_name, (death_age - birth_yr) AS age
FROM president
ORDER BY age DESC
LIMIT 1;
```

3. List 3 hobbies that presidents have in common

```
JavaScript
db.pres_hobby.aggregate([
  {
    $group: {
      _id: "$hobby",
      num_presidents: { $sum: 1 }
    }
  },
  {
    $match: {
      num_presidents: { $gte: 3 }
    }
  }
]);
```

Equivalent SQL code:

```
Unset
SELECT hobby, COUNT(*) AS num_president
FROM pres_hobby
GROUP BY hobby
HAVING COUNT(*) >= 3;
```

4. list the details for every president along with the votes they got each election year

```javascript
db.president.aggregate([
 {
  $lookup: {
   from: "election",
   localField: "pres_name",
   foreignField: "candidate",
   as: "elections"
  }
 },
 {
  $unwind: "$elections"
 },
 {
  $project: {
   _id: 0,
   pres_name: 1,
   birthyear: 1,
   yearserved: 1,
   deathage: 1,
   partyage: 1,
   stateborn: 1,
   election_year: "$elections.election_year",
   votes: "$elections.votes"
  }
 }
]);
```

SQL equivalent:

```
Unset
SELECT p.pres_name,
       p.birthyear,
       p.yearserved,
       p.deathage,
       p.partyage,
       p.stateborn,
       e.election_year,
       e.votes
FROM president p
LEFT JOIN election e ON p.pres_name = e.candidate;
```

5.list the name of the youngest wife

```
JavaScript

db.pres_marriage.aggregate([
  {
    $sort: { sp_age: 1 }
  },
  {
    $limit: 1
  },
  {
    $project: {
      _id: 0,
      youngest_wife: "$sp_name"
    }
  }
]);
```

SQL equivalent:

```
Unset
SELECT sp_name
FROM pres_marriage
ORDER BY sp_age
LIMIT 1;
```

In terms of querying ability, MongoDB and SQL databases diverge notably in their respective query languages and join capabilities. MongoDB employs a JSON-like query language called MongoDB Query Language (MQL), which offers operators like `$eq`, `$gt`, `$lt`, `$in`, `$and`, and `$or` for crafting complex queries. This provides flexibility in querying data stored in JSON-like documents.

On the other hand, SQL databases utilize Structured Query Language (SQL), providing a declarative syntax with statements such as SELECT, INSERT, UPDATE, DELETE, and JOIN. SQL's syntax is optimized for querying tabular data, offering powerful features like joins, which allow for efficient combination of data from multiple tables. While MongoDB supports limited join operations through the `$lookup` aggregation stage, these operations are generally less efficient compared to SQL databases, particularly when handling large datasets. Thus, while MongoDB excels in querying document-based data structures with its flexible MQL, SQL databases stand out in their efficient handling of relational data through SQL's robust querying and join capabilities.

## Conclusion

MongoDB is one of the most popular open-source non-relational DBMS in the field, offering a capable combination of flexibility, scalability, and performance. MongoDB's document-oriented data model provides a natural and intuitive way to represent complex and different data, helping developers build agile and adaptable applications.

One of MongoDB's key strengths lies in its support for schema validation, which enables developers to enforce integrity constraints and maintain data consistency while enjoying the benefits of a schema-less approach. By allowing the definition of validation rules at the collection level, MongoDB provides detailed and strong control over data integrity, ensuring that only valid and well-structured documents are stored in the database.

Furthermore, MongoDB's query language, MQL, offers powerful and efficient data manipulation and retrieval operations for the database structure. From simple CRUD operations to complex aggregation pipelines, MQL supports developers to interact with data flexibly and intuitively, bringing out the full capabilities of MongoDB's document-based model.

The examples presented in this report, drawn from a hypothetical "Presidential Database," demonstrate MongoDB's MQL power and highlight its ability to handle different cases. MongoDB's querying capabilities rival those of traditional SQL databases, offering a strong alternative for developers seeking a more flexible and scalable solution.

In the modern day, MongoDB plays a pivotal role in supporting innovations and organizations in storing and manipulating data. By utilizing MongoDB's strengths in data modeling, schema validation, and query optimization, developers can build robust applications capable of surviving in the modern business.

Reference:

*MongoDB - query document*. Tutorialspoint. (n.d.).
https://www.tutorialspoint.com/mongodb/mongodb_query_document.htm

*Atlas stream processing*
Atlas Stream Processing Overview - MongoDB Atlas. (n.d.).
https://www.mongodb.com/docs/atlas/atlas-sp/overview/#std-label-atlas-sp-vali
dation

*Data modeling*
Data Modeling - MongoDB Manual. (n.d.).
https://www.mongodb.com/docs/manual/data-modeling/

MongoDB query API. (n.d.).
https://www.w3schools.com/mongodb/mongodb_query_api.php