NoSQL Concepts: Documents vs Tables, Collections, BSON (with MongoDB)

1. Problem Statement

Traditional relational databases use tables and strict schemas, which can limit flexibility and scalability for modern, rapidly changing applications. NoSQL databases like MongoDB offer a different approach, using documents, collections, and flexible data structures.

The challenge:
How do you model, store, and retrieve data efficiently in MongoDB? What are the differences between databases, collections, and documents, and how does BSON enable flexible, high-performance data storage?

---

2. Learning Objectives

By the end of this lesson, you will be able to:

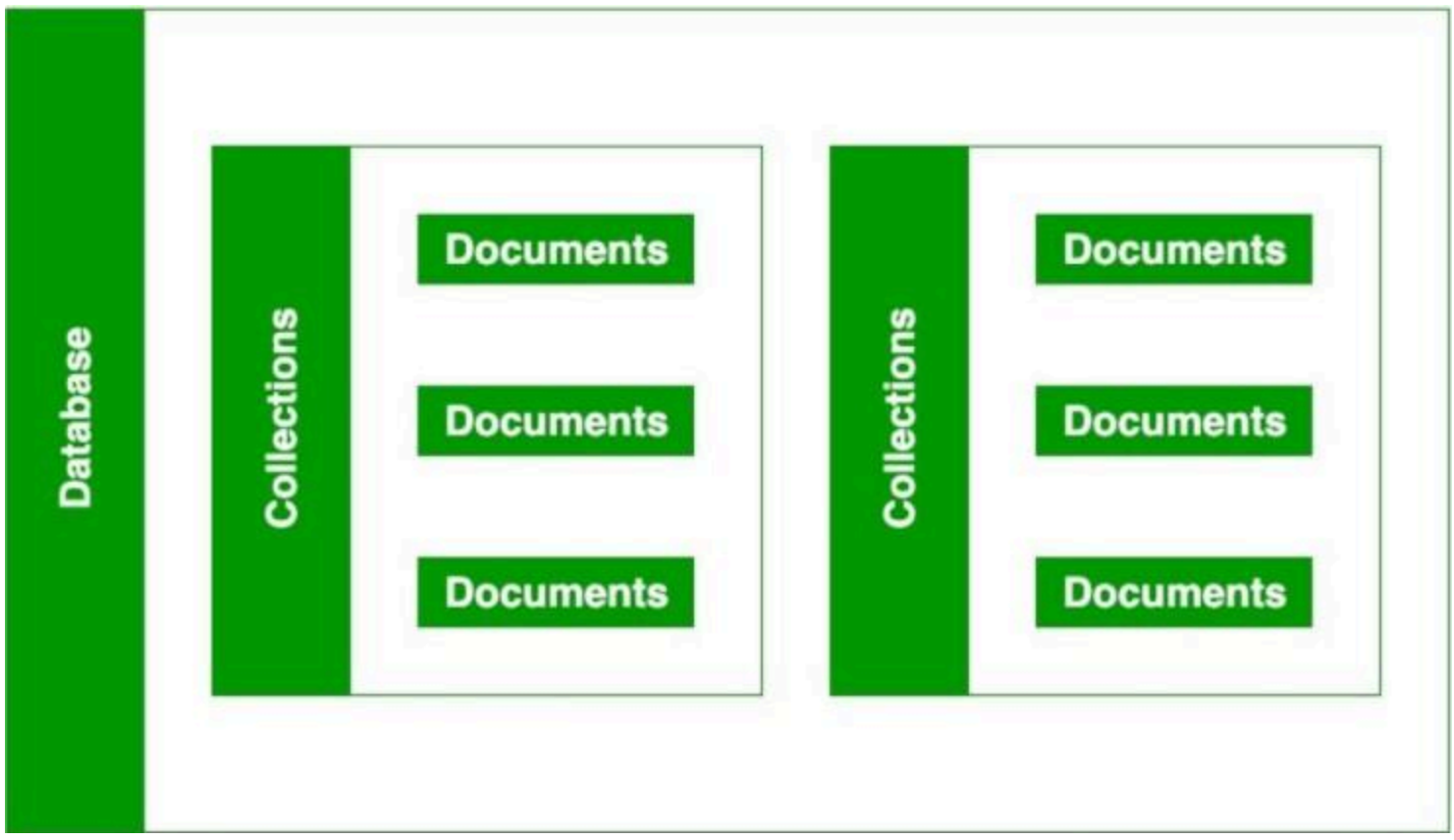
- Understand the core building blocks of MongoDB: databases, collections, and documents.
- Compare MongoDB's document-based model with relational tables.
- Explain the purpose and structure of BSON in MongoDB.
- Apply naming conventions and restrictions for databases, collections, and documents.
- Create and manipulate databases, collections, and documents using MongoDB commands.
- Recognize best practices for modeling data in MongoDB.

---

3. Concept Introduction with Analogy

Key Differences Between Databases, Collections, and Documents

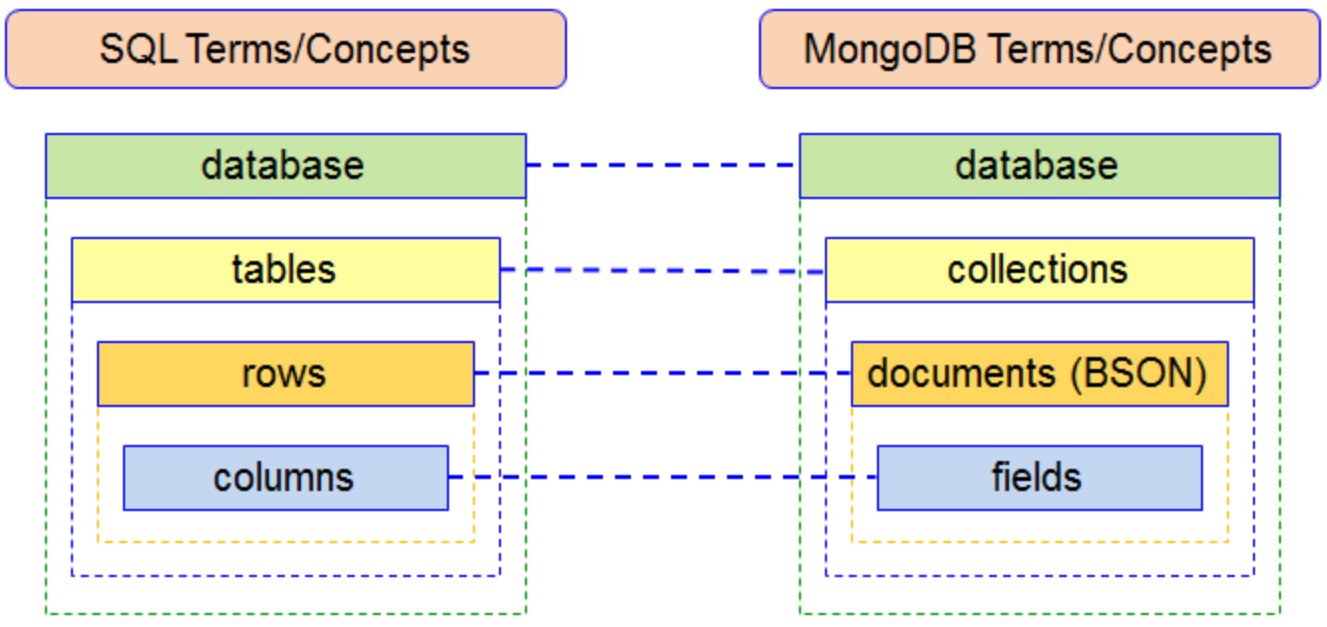
- **Database**: A container for collections, providing structure and logical isolation for data.
- **Collection**: A group of documents within a database, similar to a table in relational databases.
- **Document**: A single data record within a collection, stored as a BSON object.



Analogy: Filing Cabinets, Folders, and Documents

- **Database:** Like a filing cabinet, it contains multiple folders (collections) and keeps everything organized for a specific purpose (e.g., HR, Library).
- **Collection:** Each folder inside the cabinet holds related documents, but unlike relational tables, each document can look different.
- **Document:** A document is like a sheet of paper inside a folder, with information written in any structure you need. You can add or remove fields as needed, similar to writing notes on a paper.

Relational Table vs. MongoDB Document:
A table is like a spreadsheet with fixed columns, while a MongoDB document is a flexible form where each row (document) can have different fields.



4. Technical Deep Dive

Databases in MongoDB

- Top-level container for data.
- Can have multiple databases per server, each logically isolated.
- Created when you insert data (not just by naming).
- Naming restrictions: case-insensitive, <64 chars, no special characters like `/ . "$*|?`.

Example:

```
use LibraryDB
```

Collections in MongoDB

- Similar to tables in relational databases.
- Hold groups of documents, but schema is flexible (schema-less).
- Multiple collections per database.
- Naming restrictions: start with letter/_; no $, empty string, null char, or prefix `system.` ; max length 120 bytes.

Example:

```
db.books.insertOne({ title: "Learn MongoDB", author: "Jane Doe", year: 2023 })
```

Documents in MongoDB

- Core data record, stored as BSON (Binary JSON).
- Flexible: Each document can have different fields.
- Each document must have a unique `_id` field (like a primary key).
- Field naming restrictions: must be strings, no null chars, top-level fields not starting with `$` .
- Max document size: 16MB.

Example:

```
{
  "_id": ObjectId("..."),
  "title": "MongoDB Basics",
  "author": "John Doe",
  "year": 2025
}
```

The _id Field

- Every document must have a unique `_id` .
- If not provided, MongoDB generates an ObjectId.
- ObjectId: 12 bytes (timestamp, random value, counter), roughly ordered by creation time.

BSON: Binary JSON

- BSON extends JSON with more data types (e.g., binary, date, decimal128).
- Enables efficient storage and retrieval.
- Key types: double, string, object, array, binData, objectId, bool, date, null, regex, int, long, decimal, etc.

Example:

```
db.books.insertOne({ title: "Learn MongoDB", price: NumberDecimal("19.99") })
```

Decimal128 and Precision

- For high-precision numbers (e.g., financial data), use decimal128.
- Example: `NumberDecimal("9823.1297")`
- Avoids floating-point rounding errors.

Comparing MongoDB to Relational Databases

Relational DB	MongoDB
Database	Database
Table	Collection
Row	Document
Column	Field

---

5. Step-by-Step Data Modeling & Code Walkthrough

1. Create a Database

```
use LibraryDB
```

2. Create a Collection and Insert a Document

```
db.books.insertOne({
  title: "MongoDB for Beginners",
  author: "Alice Johnson",
  year: 2023
})
```

3. View Documents

```
db.books.find()
```

4. Custom _id Example

```
db.books.insertOne({
  _id: "custom123",
  title: "Advanced MongoDB",
  author: "Bob Smith",
  year: 2024
})
```

5. Using Decimal128

```
db.products.insertOne({
  name: "Gold Bar",
  price: NumberDecimal("12345.6789")
})
```

---

6. Interactive Challenge

Your Turn!

- Create a new database called `ShopDB`.
- Add a collection named `products`.
- Insert three product documents, each with different fields (e.g., one with price, one with stock, one with a nested object).
- Retrieve all products and observe the flexible schema.
- Insert a product with a custom `_id` and another with a decimal128 price.
- Try inserting a document larger than 16MB (expect an error).
- Query products by type using the `$type` operator.

---

7. Common Pitfalls & Best Practices

- **Use Descriptive Names:** Name databases and collections for clarity.
- **Avoid Special Characters:** Stick to allowed characters in names.
- **Design Efficient Documents:** Use nested structures and arrays wisely; avoid excessively large documents.
- **Leverage BSON Types:** Use decimal128 for financial data, ObjectId for unique identifiers.
- **Keep _id Unique:** Ensure no duplicate `_id` values in a collection.
- **Remember Document Size Limit:** Use GridFS for very large files.

---

8. Quick Recap & Key Takeaways

- MongoDB uses databases, collections, and documents (BSON) instead of tables and rows.
- Collections are schema-less, allowing flexible data models.
- Each document must have a unique `_id`.
- BSON supports more data types than JSON, enabling rich data modeling.
- Use decimal128 for precise calculations and ObjectId for unique, ordered IDs.
