Document Databases

A **document database** is a type of non-relational database that stores data as structured documents, typically in **JSON** (JavaScript Object Notation) format. Document databases are designed to be simple, flexible, and scalable, making them ideal for applications that require dynamic schemas or hierarchical data structures.

For example, an order document in a document database might look like this:

```json
{
  "orderno": "748745375",
  "date": "June 30, 2088 1:54:23 AM",
  "trackingno": "TN0039291",
  "custid": "11045",
  "customer": {
    "custid": "11045",
    "fname": "Sue",
    "lname": "Hatfield",
    "address": "1409 Silver Street",
    "city": "Ashland",
    "state": "NE",
    "zip": "68003"
  }
}
```

In this example, the order document contains nested information about the customer, which is a natural way to represent hierarchical data. Unlike relational databases, document databases do not require a predefined schema, allowing for greater flexibility in how data is stored.

What is JSON?

JSON (JavaScript Object Notation) is a lightweight data-interchange format that is easy for humans to read and write, and easy for machines to parse and generate. JSON is built on two universal data structures:

1. **A collection of name/value pairs**: This is often represented as an object, record, struct, dictionary, hash table, or associative array in various programming languages.
2. **An ordered list of values**: This is typically represented as an array, vector, list, or sequence in most programming languages.

These two structures are supported by virtually all modern programming languages, making JSON an excellent choice for data interchange.

JSON Syntax

JSON syntax is straightforward and consists of the following elements:

- **Objects**: Enclosed in curly braces `{}`, containing key-value pairs.
- **Arrays**: Enclosed in square brackets `[]`, containing ordered lists of values.
- **Values**: Can be strings, numbers, objects, arrays, `true`, `false`, or `null`.

For example:

```json
{
  "name": "John",
  "age": 30,
  "isStudent": false,
  "courses": ["Math", "Science"],
  "address": {
    "street": "123 Main St",
    "city": "New York"
  }
}
```

Binary JSON (BSON)

BSON (Binary JSON) is a binary-encoded serialization of JSON-like documents. It extends JSON to support additional data types, such as dates and binary data, and is designed to be lightweight, traversable, and efficient. BSON is used by MongoDB to store and retrieve documents.

For example, the JSON object `{"hello": "world"}` is encoded in BSON as:

```
\x16\x00\x00\x00  // Total document size
\x02              // Type String
hello\x00         // Field name
\x06\x00\x00\x00world\x00  // Field value
```

```
\x00              // End of object
```

XML (Extensible Markup Language)

XML (Extensible Markup Language) was a precursor to JSON as a data interchange format. XML is structurally similar to HTML but allows for custom tags, making it highly flexible. XML was widely used for web pages, separating content (XML) from formatting (CSS).

An example of XML:

```xml
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
</CATALOG>
```

XML-Related Tools and Technologies

Several tools and technologies are associated with XML:

- **XPath**: A syntax for retrieving specific elements from an XML document.
- **XQuery**: A query language for interrogating XML documents, often referred to as the "SQL of XML."

- **DTD (Document Type Definition)**: A language for describing the allowed structure of an XML document.
- **XSLT (Extensible Stylesheet Language Transformations)**: A tool for transforming XML into other formats, including non-XML formats like HTML.

Why Document Databases?

Document databases address the **impedance mismatch** problem that arises when trying to persist complex objects from object-oriented (OO) systems in relational databases. In OO programming, objects often have complex relationships through inheritance and composition, which do not map neatly to the tabular structure of relational databases.

Document databases, on the other hand, store data in a self-describing format (like JSON or BSON), which aligns well with modern applications that use JSON or XML as a transport layer. This makes it easier to store and retrieve complex, hierarchical data without the need for extensive deconstruction and reconstruction.

MongoDB

MongoDB is one of the most popular document databases. It was developed in 2007 by veterans of DoubleClick, who recognized the limitations of relational databases for handling large-scale, high-throughput applications like ad serving. MongoDB was designed to handle massive amounts of data and high query loads, making it suitable for modern web applications.

- **MongoDB Atlas**: A fully managed MongoDB service in the cloud (DBaaS).
- **MongoDB Enterprise**: A subscription-based, self-managed version of MongoDB.
- **MongoDB Community**: A free, open-source version of MongoDB for developers.

MongoDB Structure

MongoDB organizes data into **databases**, which contain **collections**. Each collection contains **documents**, which are JSON-like structures. Unlike relational databases, MongoDB does not require a predefined schema, so documents within the same collection can have different structures.

For example, a collection might contain the following documents:

```json
```

```
{ "name": "will", "eyes": "blue", "birthplace": "NY" }
{ "name": "Jeff", "eyes": "blue", "loc": [40.7, 73.4] }
{ "name": "brendan", "aliases": ["el diablo"] }
{ "name": "matt", "pizza": "DiGiorno", "height": 72 }
```

Relational vs. MongoDB/Document DB

Here's a comparison of key concepts between relational databases and MongoDB:

- **RDBMS Database** → **MongoDB Database**
- **RDBMS Table/View** → **MongoDB Collection**
- **RDBMS Row** → **MongoDB Document**
- **RDBMS Column** → **MongoDB Field**
- **RDBMS Index** → **MongoDB Index**
- **RDBMS Join** → **MongoDB Embedded Document**
- **RDBMS Foreign Key** → **MongoDB Reference**

MongoDB Features

MongoDB offers several powerful features:

- **Rich Query Support**: MongoDB supports a wide range of CRUD (Create, Read, Update, Delete) operations.
- **Indexing**: MongoDB supports primary and secondary indexes on document fields, enabling fast query performance.
- **Replication**: MongoDB supports replica sets with automatic failover, ensuring high availability.
- **Load Balancing**: Built-in load balancing ensures that queries are distributed evenly across the database.

Interacting with MongoDB

There are several tools for interacting with MongoDB:

- **mongosh**: The MongoDB Shell, a command-line interface for interacting with MongoDB.
- **MongoDB Compass**: A free, open-source GUI for working with MongoDB databases.
- **DataGrip**: A third-party tool for database management.

- **Programming Libraries**: MongoDB provides libraries for most major programming languages, including **PyMongo** (Python) and **Mongoose** (JavaScript/Node.js).

MongoDB in Docker

MongoDB can be run in a Docker container, making it easy to set up and manage. To create a MongoDB container:

1. Map the host port to the container port (default is 27017).
2. Set the initial username and password for the superuser.
3. Optionally, configure environment variables for the root username and password.

MongoDB Compass

MongoDB Compass is a GUI tool for interacting with MongoDB instances. It allows you to:

- Create and manage databases and collections.
- Import and export data.
- Run queries and view results in a user-friendly interface.

MongoDB Queries

MongoDB queries are similar to SQL queries but use a different syntax. For example:

- **Find all users**:
  ```javascript
  db.users.find()
  ```
- **Find users with a specific name**:
  ```javascript
  db.users.find({ "name": "Davos Seaworth" })
  ```
- **Find movies with a rating of PG or PG-13**:
  ```javascript
  db.movies.find({ rated: { $in: ["PG", "PG-13"] } })
  ```

Comparison Operators

MongoDB supports a variety of comparison operators for querying:

- **$eq**: Matches values equal to a specified value.
- **$gt**: Matches values greater than a specified value.
- **$gte**: Matches values greater than or equal to a specified value.
- **$in**: Matches any of the values specified in an array.
- **$lt**: Matches values less than a specified value.
- **$lte**: Matches values less than or equal to a specified value.
- **$ne**: Matches all values not equal to a specified value.
- **$nin**: Matches none of the values specified in an array.

PyMongo

PyMongo is the official Python library for interacting with MongoDB. Here's how to use it:

- **Connecting to MongoDB**:
  ```python
  from pymongo import MongoClient
  client = MongoClient('mongodb://user_name:pw@localhost:27017')
  ```

- **Inserting a Document**:
  ```python
  db = client['ds4300']
  collection = db['myCollection']
  post = { "author": "Mark", "text": "MongoDB is Cool!", "tags": ["mongodb", "python"] }
  post_id = collection.insert_one(post).inserted_id
  print(post_id)
  ```

- **Counting Documents**:
  ```python
  count = db.collection.count_documents({})
  print(count)
  ```