DS 4300: Document Databases and Mongo DB

Document Database

- **Document Database** = Non-relational database that stores data as structured documents usually in JSON
- Designed to be simple, flexible, and scalable
- Example:

```
{
  "_id": "user_001",
  "name": "John Doe",
  "email": "john.doe@example.com",
  "interests": ["Tech", "Travel"],
  "posts": [
    {
      "post_id": "post_1001",
      "content": "Learning NoSQL!",
      "likes": 15
    },
    {
      "post_id": "post_1002",
      "content": "Travel tips for Japan?",
      "likes": 42
    }
  ]
}
```

What is JSON?

- ***JSON (JavaScript Object Notation)***
  - Lightweight data-interchange format
  - Easy for humans to read and write
  - Easy for machines to parse and generate
- Built on 2 structures
  - A collection of name/value pairs. In various languages, this is operationalized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
  - An ordered list of values. In most languages, this is operationalized as an array, vector, list, or sequence.
- These are two universal data structures supported by virtually all modern programming languages
  - Thus, JSON makes a great data interchange format.
- ***Syntax***:
  - *Object*: { [whitespace] "string" [whitespace] : value [whitespace] [, repeat] }
  - *Array*: [[whitespace] value [, repeat]]
  - *Value*: [whitespace] (string | number | object | array | true | false | null) [whitespace]

Binary JSON

- BSON -> Binary JSON

- Binary-encoded serialization of a JSON-like document structure
- Supports extended types not part of basic JSON (e.g. Date, BinaryDate, etc)
- ***Lightweight*** = keep space overhead to a minimum
- ***Traversable*** = designed to be easily traversed, which is vitally important to a document DB
- ***Efficient*** = encoding and decoding must be efficient
- Supported by many programming languages

```
{"hello": "world"} →
\x16\x00\x00\x00                // total document size
\x02                            // 0x02 = type String
hello\x00                       // field name
\x06\x00\x00\x00world\x00       // field value
\x00                            // 0x00 = type EOO ('end of object')
```

XML (eXtensible Markup Language)

- Precursor to JSON as data exchange format
- XML + CSS -> web pages that separated content and formatting
- Structurally similar to HTML, but tag set is extensible
- ***Related Tools/ Technologies***
  - **Xpath** = syntax for retrieving specific elements from an XML doc
  - **Xquery** = query language for interrogating XML
  - **DTD** = Document Type Definition - A language for describing the allowed structure of an XML document
  - **XSLT** = eXtensible Stylesheet Language Transformation- tool to transform XML into other formats, including non-XML formats such as HTML

Why document databases?

- Document databases address the impedance mismatch problem between object persistence in OO systems and how relational DBs structure data.
  - OO programming -> inheritance and composition of types.
  - How to save a complex object to a relational database -> deconstruct it
- Structure of a document is self-describing
- Well-aligned with apps that use JSON/XML as a transport layer.

MongoDB

- Started in 2007 after Doubleclick was acquired by Google, and 3 of its veterans realized the limitations of relational databases for serving > 400,000 ads per second
- MongoDB was short for Humongous Database
- MongoDB Atlas released in 2016 → documentdb as a service

MongoDB Structure

Collection A: document 1, document 2, document 3
Collection B: document 1, document 2, document 3
Collection C: document 1, document 2, document 3

MongoDB Documents

- No predefined schema for documents is needed
- Every document in a collection could have different data/schema

Relational vs Mongo/Document Databases

- ***RDBMS***
    - Database
    - Table/view
    - Row
    - Column
    - Index
    - Join
    - Foreign key
- ***MongoDB***
    - Database
    - Collection
    - Document
    - Field
    - Index
    - Embedded document
    - Reference

MongoDB Features

- Rich Query Support -> Robust support for all CRUD ops
- Indexing -> Supports primary and secondary indices on document fields
- Replication -> Supports replica sets with automatic failover
- Built-in load balancing

MongoDB Versions

- MongoDB Atlas = Fully managed MongoDB service in the cloud (DBaaS)
- MongoDB Enterprise = Subscription-based, self-managed version of MongoDB
- MongoDB Community = Source-available, free-to-use, self-managed

Interacting with MongoDB

- **mongosh** → MongoDB Shell
    - CLI tool for interacting with a MongoDB instance
- **MongoDB Compass** = free, open-source GUI to work with a MongoDB database
- DataGrip and other 3rd Party Tools
- Every major language has a library to interface with MongoDB
    - PyMongo (Python), Mongoose (JavaScript/node), …

Mongo Syntax: find()

- find(...) is like SELECT
    - collection.find({ *filters* }, { *project* })
- SELECT *FROM users;
    - Use mflix
    - db.users.find()
- SELECT* FROM users WHERE name = “Davos Seaworth”;
    - db.users.find({“name”: “Davos Seaworth”})
- SELECT * FROM movies WHERE rated in ("PG", "PG-13")
    - db.movies.find({rated: {$in:["PG", "PG-13"]}})
- Return movies which were released in Mexico and have an IMDB rating of at least 7

- db.movies.find({
  "countries": "Mexico",
  "imdb.rating": { $gte: 7 }
  })

Mongosh - project

- Return movies from the movies collection which were released in 2010 and either won at least 5 awards or have a genre of Drama
  - db.movies.find({
    “year”: 2010,
    $or: [
    { "awards.wins": { $gte: 5 } },
    { “genres”: "Drama" }] })
- ***Comparison Operators:***

  Equality (`$eq`)

  - Matches documents where the field is equal to the specified value.
  - Example:
    query = {"age": {"$eq": 25}}

  Inequality (`$ne`)

  - Matches documents where the field is not equal to the specified value.
  - Example:
    query = {"status": {"$ne": "inactive"}}

  Greater Than (`$gt`)

  - Matches documents where the field is greater than the specified value.
  - Example:
    query = {"price": {"$gt": 100}}

  Greater Than or Equal (`$gte`)

  - Matches documents where the field is greater than or equal to the specified value.
  - Example:
    query = {"rating": {"$gte": 4.5}}

  Less Than (`$lt`)

  - Matches documents where the field is less than the specified value.
  - Example:
    query = {"stock": {"$lt": 50}}

  Less Than or Equal (`$lte`)

- Matches documents where the field is less than or equal to the specified value.
- Example:
  query = {"temperature": {"$lte": 0}}

In (`$in`)

- Matches documents where the field value is in the specified array.
- Example:
  query = {"category": {"$in": ["electronics", "furniture"]}}

Not In (`$nin`)

- Matches documents where the field value is not in the specified array.
- Example: query = {"role": {"$nin": ["admin", "moderator"]}}

Mongosh - countDocuments()

- How many movies from the movies collection were released in 2010 and either won at least 5 awards or have a genre of Drama
  - db.movies.countDocuments({
    "year": 2010,
    $or: [
    { "awards.wins": { $gte: 5 } },
    { "genres": "Drama" }
    ]
    })

PyMongo

- ***Python library for interfacing with MongoDB instances***

```
from pymongo import MongoClient
client = MongoClient(
      'mongodb://user_name:pw@localhost:27017'
)
```

- *Getting a database and a collection*

```
from pymongo import MongoClient
client = MongoClient(
      'mongodb://user_name:pw@localhost:27017'
)

db = client['ds4300']
collection = db['myCollection']
```

- *Inserting single document*

```
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)
```

- *Count documents in collection*
    - SELECT count(*) FROM collection
      `demodb.collection.count_documents({})`