Document DBs & Mongo

Created	@February 10, 2025 9:22 AM
Class	DS 4300
Reviewed	☐

Document DBs

Document Types

JSON

- JavaScript Object Notation
- JSON is built on two structures
  - a collection of name/value pairs
  - an ordered list of values
- Two universal data structures supported by virtually all modern programming languages
  - Makes JSON great data interchange format

BSON

- binary JSON
- binary-encoded serialization of a JSON-like document
- supports types not part of basic JSON
- lightweight - keep space overhead to min
- traversable - designed to be easily traversed, which is vital to document db
- efficient - encoding and decoding must be efficient

XML

- precursor to JSON
- XML + CSS → web pages that separated content and formatting
- Structurally similar to HTML, but tag set is extensible

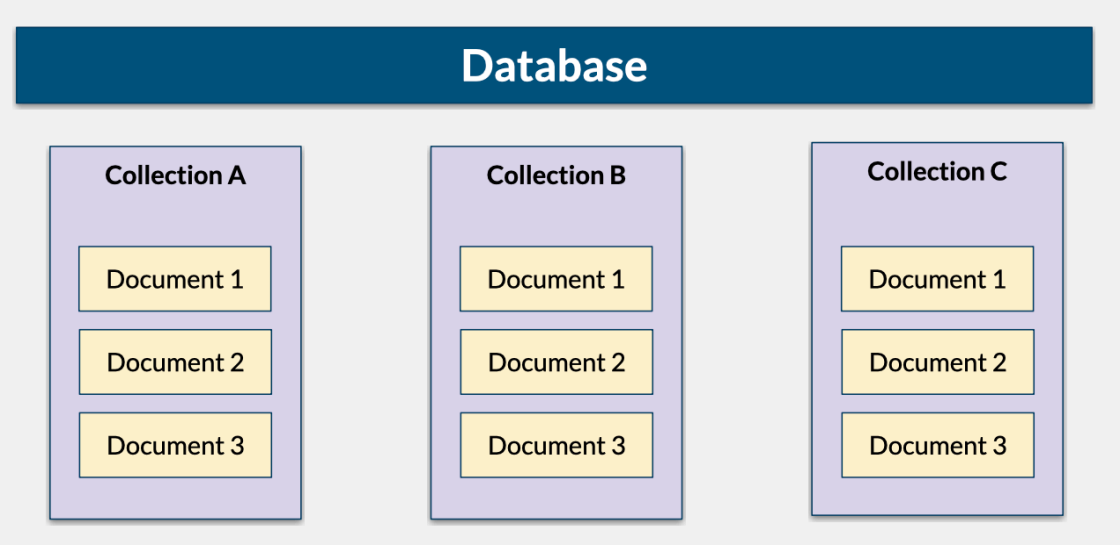
Why Document DB?

- address the impedance mismatch problem between persistence in OO systems and how relational DBs structure data
  - OO programming → inheritance and composition of types
  - How do we save a complex object to a relational database?
    - Basically have to deconstruct it
- Structure of doc is self-describing

MongoDB

- MongoDB → short for humongous database

Structure



- database is a series of collections of documents
- Documents:
  - No predefined schema for documents is needed
  - every document in a collection could have different data/schema
- Relational vs. Mongo/Document DB

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

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
- load balancing built in
- Interacting with MongoDB
  - mongosh → MongoDB Shell
  - 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 (JS/node)

▼ Commands

```
# find is like select
collection.find({ ___ }, { ____ })
                  #filters.       #projections
# SELECT * FROM USERS
use mflix
db.users.find()

#SELECT *
#FROM years
#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 }
 } )

#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 Ops

Name	Description
`$eq`	Matches values that are equal to a specified value.
`$gt`	Matches values that are greater than a specified value.
`$gte`	Matches values that are greater than or equal to a specified value.
`$in`	Matches any of the values specified in an array.
`$lt`	Matches values that are less than a specified value.
`$lte`	Matches values that are less than or equal to a specified value.
`$ne`	Matches all values that are not equal to a specified value.
`$nin`	Matches none of the values specified in an array.

-

PyMongo

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

- You can copy connection string in Compass

```
db = client['ds4300'] # or client.ds4300
collection = db['myCollection'] #or 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)
```

▼ Find All Movies from 2000

```
from bson.json_util import dumps

# Find all movies released in 2000
movies_2000 = db.movies.find({"year": 2000})

# Print results
for movie in movies_2000:
    print(dumps(movie,indent = 2))
```

- Containerization:

- Can use containers to do horizontal scaling
- Image is blueprint for creating identical containers
- Sometimes projects need more than one container (think CS3200, api, db, flask)
- Ports: first is host, second is container port