Document Databases & MongoDB

Detailed Description (Cover Image as Text):

- The background typically features a clean layout with the Northeastern University logo or branding (depending on how the slide was designed).
- The text is centered, reading “Document Databases & MongoDB” in a larger font.
- The subtitle “DS 5300” indicates the course code.
- Below, the instructor’s name, “Mark Fontanot, PhD,” is listed, along with “Northeastern University.”

This cover page sets the stage for a presentation about modern document databases—particularly MongoDB—and how they differ from or complement traditional relational databases.

Slide 2: Introduction to Document Databases

Key Points:

- A **Document Database** is a type of database that stores data as **semi-structured documents**, usually in **JSON** format.
- These databases are designed to be **simpler**, **flexible**, and **scalable**.

Expanded Explanation:

- **Semi-structured** means that the data does not strictly follow a fixed schema like relational tables do. Instead, each “document” can contain varying sets of fields and nested structures.
- **JSON** (JavaScript Object Notation) is a lightweight data-interchange format that is human-readable and widely used in modern web applications.
- Document databases allow you to store data in a way that often more closely matches the objects in your application’s code, leading to less “impedance mismatch” compared to traditional relational models.
- “Simpler” often refers to fewer steps when mapping objects from code to database documents, while “flexible” and “scalable” refer to the ease of horizontal scaling and handling evolving data structures.

Slide 3: What Is JSON?

Key Points:

- **JSON (JavaScript Object Notation)** is a lightweight, text-based data interchange format.

- It is language-independent but uses conventions familiar to programmers of the C-family of languages (C, C++, Java, JavaScript, etc.).
- JSON is easy for humans to read and write, and easy for machines to parse and generate.
- It is widely used by modern applications, web APIs, and configurations.

Expanded Explanation:
- JSON typically represents data in **key-value pairs**.
- Example structure:

```
{
  "name": "Alice",
  "age": 30,
  "skills": ["Python", "Databases", "Machine Learning"]
}
```

- JSON has become a de facto standard for RESTful APIs, configuration files, and data storage in NoSQL/document-oriented databases.

Slide 4: JSON Syntax

Key Points:
- **Objects** are delimited by curly braces { }.
- **Arrays** are delimited by square brackets [ ].
- **Key-value pairs** use the format "key": value, and strings must be in double quotes.
- Values can be strings, numbers, objects, arrays, booleans, or null.

Expanded Explanation:
- Proper formatting and strict rules around quotes and commas are important for valid JSON.
- Missing commas, incorrect bracket usage, or single quotes instead of double quotes can break JSON.
- JSON data can nest objects within objects and arrays, enabling complex hierarchies.

Slide 5: Binary JSON (BSON)

Key Points:

- **BSON** stands for **Binary JSON**.
- MongoDB uses BSON internally as its storage format.
- BSON extends the JSON model with additional data types (e.g., for dates, binary data) and is more efficient for machine processing.

Expanded Explanation:

- BSON is designed to be lightweight and traversable. It encodes length information at the beginning of objects and arrays, making it easier and faster to parse.
- Although MongoDB "speaks" JSON to external clients, the actual storage on disk is in BSON, which supports features like 32-bit/64-bit integers, floating-point numbers, timestamps, etc.

Slide 6: XML (Extensible Markup Language)

Key Points:

- XML was a precursor to JSON as an exchange format.
- XML separates data from presentation and uses tags for structure.
- It is highly extensible, but can be verbose compared to JSON.

Expanded Explanation:

- **XML** stands for **Extensible Markup Language** and was widely used for data exchange in the early days of web services (SOAP, RSS, etc.).
- While still used in many enterprise environments, XML has largely been supplanted by JSON in many modern REST APIs due to JSON's simpler syntax.
- XML's flexibility lies in the ability to define custom tags, but this can also lead to more complex parsing logic.

Slide 7: XML-Related Tools/Technologies

Key Points:

- **XPath**: A query language for selecting nodes from an XML document.
- **XQuery**: A functional query language that can query and transform XML data.
- **XSLT**: A language for transforming XML documents into other formats.

Expanded Explanation:

- These tools can filter, query, and restructure XML data.
- While powerful, these technologies can be more cumbersome compared to the simpler query capabilities of JSON-based document databases.

Slide 8: Why Document Databases?

Key Points:

- Document databases are useful when data does not fit well into a tabular schema or when the schema is constantly evolving.
- They can handle large volumes of data and support high throughput (e.g., thousands of requests per second).
- They are often easier to scale horizontally across multiple servers (sharding).

Expanded Explanation:

- In modern web applications, data often changes shape rapidly. For instance, a user profile might gain new fields as features evolve. Document databases make these schema changes straightforward.
- High scalability and flexible schemas make document databases a good choice for real-time analytics, mobile apps, content management systems, and more.

Slide 9: MongoDB – Introduction

Key Points:

- **MongoDB** is one of the most popular document-oriented databases.
- Known for storing data in JSON-like documents (internally BSON).
- Provides high availability, horizontal scaling, and robust query capabilities.

Expanded Explanation:

- MongoDB was developed by a company formerly known as 10gen; the database quickly became a leader in the NoSQL movement.
- It has a flexible schema design, meaning each document in a collection can have a unique structure.

Slide 10: MongoDB Background

Key Points:

- Started in 2007 after DoubleClick was acquired by Google; engineers realized relational databases were hitting limitations with massive ad-serving workloads (~400,000 ads per second).

- The name “Mongo” is short for “Humongous,” reflecting its design for very large data volumes.
- **MongoDB Atlas** was released in 2016, offering a cloud-hosted DBaaS (Database as a Service).

Expanded Explanation:

- MongoDB’s founders sought a more scalable solution than traditional relational databases could provide at the time.
- MongoDB, Inc. (formerly 10gen) offers both a community edition (open source) and enterprise versions with additional features.
- The success of Atlas shows how important managed cloud services have become, as it simplifies deployment and scaling.

Slide 11: MongoDB Structure

Key Points:

- A **MongoDB** database consists of one or more **collections**.
- A **collection** is analogous to a table in relational databases, but without a fixed schema.
- Each **collection** contains multiple **documents** (analogous to rows).

Expanded Explanation:

- **Database** > **Collection** > **Document** is the hierarchy.
- There is no enforced schema at the collection level, meaning each document can have different fields.
- This flexibility allows for faster development cycles and easier updates to data models.

Slide 12: MongoDB Documents

Key Points:

- There is no predefined schema for documents.
- Every document in a collection can have different data or schema.
- Documents are typically stored as BSON, but displayed as JSON when retrieved.

Expanded Explanation:

- A single collection could store user profiles that differ in the number of fields. For example, some users might have a “hobbies” field, while others do not.

- The lack of a predefined schema can be a double-edged sword: it offers flexibility but requires careful handling of inconsistent data.

Slide 13: Relational vs. Mongo/Document DB

Key Points:

- Relational databases store data in tables with rows and columns, enforcing a strict schema.
- Document databases store data in collections of flexible JSON-like documents.
- Each approach has advantages: relational excels at complex queries with joins and transactional integrity, while document databases excel at flexibility, speed, and horizontal scalability.

Expanded Explanation:

- **Relational**: Best for scenarios requiring strong consistency, complex joins, and ACID transactions (banking, financial records).
- **Document**: Best for fast-evolving applications, content management, analytics, or real-time data feeds where structure is not always consistent.

Slide 14: MongoDB Features

Key Points:

- **Rich Query Support**: Full CRUD (Create, Read, Update, Delete) operations.
- **Indexes**: Supports primary and secondary indexes on any field.
- **Replication**: Provides high availability with replica sets.
- **Sharding**: Enables horizontal scaling.
- **Automatic Failover**: In replica sets, one node can automatically take over if the primary fails.

Expanded Explanation:

- Queries can be as simple or as complex as needed, including aggregation pipelines that transform and analyze data in place.
- MongoDB’s replication model allows for distributed clusters that remain online even if one node goes down.
- Sharding splits data across multiple machines, balancing the load.

Slide 15: MongoDB Versions

Key Points:

- **MongoDB Atlas**: Managed MongoDB service in the cloud (DBaaS).
- **MongoDB Enterprise**: Commercial edition with advanced security and features.
- **MongoDB Community**: Free, self-managed version of MongoDB.

Expanded Explanation:

- Atlas simplifies deployment, scaling, and maintenance by handling backups, monitoring, and upgrades automatically.
- Enterprise edition includes features like encrypted storage engines, LDAP integration, and advanced auditing.
- Community edition is open source and can be installed on-premises or on your own servers in the cloud.

Slide 16: Interacting with MongoDB

Key Points:

- **mongosh** (MongoDB Shell) provides a command-line interface.
- **MongoDB Compass** is a GUI tool for visually interacting with MongoDB.
- Other drivers/tools: **PyMongo** (Python), **Mongoose** (Node.js), **DataGrip** (JetBrains), etc.

Expanded Explanation:

- The shell is useful for quick commands, maintenance, and troubleshooting.
- MongoDB Compass helps visualize collections, indexes, and run queries in a more user-friendly environment.
- Drivers in various programming languages allow developers to integrate MongoDB directly into applications.

Slide 17: MongoDB Community Edition in Docker

Key Points:

- You can run MongoDB in a container using Docker.
- Create a container, specify ports and credentials, and map local storage.
- Docker images help maintain consistent development environments.

Expanded Explanation:

- For example, using a command like:

```
docker run -d -p 27017:27017 --name mongodb -v /localpath:/data/db
mongo:latest
```

This starts a MongoDB container, maps port 27017, and mounts a local directory for data persistence.

- Docker simplifies setup and teardown of development environments.

Slide 18: MongoDB Compass

Key Points:

- A GUI tool for interacting with MongoDB.
- Allows you to visualize databases, collections, and indexes.
- You can perform CRUD operations and run queries without using the shell.

Expanded Explanation:

- MongoDB Compass is ideal for beginners or those who prefer a graphical interface.
- It provides schema analysis, data validation, and aggregation pipeline builders.

Slide 19: Load Mflix Sample Data Set

Key Points:

- In Compass, create a new database named **mflix** (a sample movie database).
- Download the **mflix sample** JSON or use built-in sample data sets.
- Import JSON files into new collections in the **mflix** database.

Expanded Explanation:

- MongoDB often provides sample datasets (e.g., sample_mflix, sample_airbnb, sample_supplies) for educational or demonstration purposes.
- Once loaded, you can explore the data in Compass or the shell, run queries, and practice.

Slide 20: Creating a Database and Collection

Key Points:

- In MongoDB, you don’t explicitly create a database until you insert data.
- You can switch to a database in the shell using use <dbName>.
- You can then insert a document into a collection, and MongoDB will create both the database and collection if they do not already exist.

Example:

```
use myDatabase
db.myCollection.insertOne({ name: "Test", value: 123 })
```

Expanded Explanation:

- This dynamic creation is different from relational databases, where you must explicitly define schemas.
- MongoDB’s flexibility reduces initial overhead but requires you to track your structure carefully.

Slide 21: mongosh – Mongo Shell: find()

Key Points:

- find() is analogous to SELECT in SQL.
- Basic usage:

```
collection.find(
  { /* filters */ },
  { /* projections */ }
)
```

- Filters define which documents to retrieve, projections define which fields to include or exclude.

Expanded Explanation:

- If you call collection.find() with no arguments, it returns all documents in the collection.
- Filters use JSON-like syntax to specify conditions, e.g., { name: "Alice" }.
- Projections let you control which fields are returned, e.g., { name: 1, _id:

0 }.

Slide 22: SQL vs. Mongo Shell – Basic “SELECT * FROM users”

SQL:

```
SELECT * FROM users;
```

MongoDB:

```
use mflix
db.users.find()
```

Expanded Explanation:

- In MongoDB, you switch to the database (use mflix) and then call db.users.find() to see all user documents.
- MongoDB doesn’t have the concept of “*” for selecting all fields, so an empty projection object or simply calling .find() without a projection will return all fields.

Slide 23: SQL vs. Mongo Shell – Basic Filter

SQL:

```
SELECT *
FROM users
WHERE name = 'Davos Seaworth';
```

MongoDB:

```
db.users.find({ name: "Davos Seaworth" })
```

Expanded Explanation:

- MongoDB uses a JSON-style filter object.
- The equality operator in MongoDB is implicit when you use a key-value pair like { name: "Davos Seaworth" }.

Slide 24: Filtering by Rated Field

SQL:

```
SELECT *
FROM movies
WHERE rated IN ('PG', 'PG-13');
```

MongoDB:

```
db.movies.find({
  rated: { $in: ["PG", "PG-13"] }
})
```

Expanded Explanation:

- $in is a MongoDB operator that checks if the field's value matches any value in the specified array.

Slide 25: Filtering by IMDB Rating >= 7

Key Points:

- Return movies with an IMDb rating of at least 7.

MongoDB Example:

```
db.movies.find(
  { "imdb.rating": { $gte: 7 } }
)
```

Expanded Explanation:

- Note how MongoDB uses dot notation ("imdb.rating") to refer to a nested field.
- $gte stands for "greater than or equal to."

Slide 26: Complex Query with Multiple Conditions

Requirement:

- Return movies from the movies collection which were released in 2010 and either:
  - Won at least 5 awards, **OR**
  - Have a genre of Drama.

MongoDB Example:

```
db.movies.find({
  $and: [
    { year: 2010 },
    {
      $or: [
        { awards: { $gte: 5 } },
        { genre: "Drama" }
      ]
    }
  ]
})
```

Expanded Explanation:

- $and and $or are logical operators.
- You can nest logical operators for more complex queries.

Slide 27: Comparison Operators Reference

Common MongoDB Operators:

- $eq (equals)
- $gt (greater than)
- $gte (greater than or equal)
- $in (in array of values)
- $lt (less than)
- $lte (less than or equal)
- $ne (not equal)
- $nin (not in array of values)

Expanded Explanation:

- These operators let you build flexible queries.
- They can be combined with logical operators ($and, $or, $not, $nor) for complex conditions.

Slide 28: mongosh – countDocuments()

Key Points:

- countDocuments() is analogous to SELECT COUNT(*) in SQL.

- Example:

```
db.movies.countDocuments({ genre: "Drama" })
```

Expanded Explanation:

- You can pass a filter object to countDocuments() to count only matching documents.
- This is typically more accurate than the older count() method because it respects the filter and uses indexes efficiently.

Slide 29: mongosh – Projection

Key Points:

- Projection specifies which fields to include (1) or exclude (0).
- Example:

```
db.movies.find(
  { year: 2010 },
  { title: 1, _id: 0 }
)
```

  - 1 means include the field, 0 means exclude.
  - _id: 0 commonly excludes the _id field.

Expanded Explanation:

- Projections can reduce bandwidth usage and clutter if you only need specific fields.
- More advanced projections can reshape documents using aggregation pipelines.

Slide 30: PyMongo

Key Points:

- **PyMongo** is the official Python library for interacting with MongoDB.

Expanded Explanation:

- It provides classes and methods to connect, query, insert, update, and delete documents from Python code.

- Great for integrating MongoDB with data science workflows in Python (e.g., using Jupyter notebooks).

Slide 31: PyMongo – Basic Usage

Connection Example:

```
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")
```

Expanded Explanation:
- MongoClient handles the connection to a local or remote MongoDB instance.
- You can pass in authentication details, replica set information, etc., as needed.

Slide 32: Getting a Database and Collection in PyMongo

Example:

```
db = client["myDatabase"]
collection = db["myCollection"]
```

Expanded Explanation:
- Once you have a client, you can access a database by name (like a dictionary key).
- You can then access a collection similarly.
- Operations on collection now directly interact with the MongoDB server.

Slide 33: Inserting a Single Document in PyMongo

Example:

```
post = {
  "title": "My First Post",
  "content": "Hello, MongoDB!",
  "tags": ["intro", "pymongo"]
```

}

```
inserted_id = collection.insert_one(post).inserted_id
print(inserted_id)
```

Expanded Explanation:

- insert_one() returns an object containing the _id of the inserted document.
- _id is a unique identifier automatically generated if not specified.
- PyMongo also supports insert_many() for bulk inserts.

Slide 34: Count Documents in a Collection (PyMongo)

Key Points:

- The PyMongo equivalent of countDocuments() can be done via the aggregation pipeline or a helper method, depending on the version of PyMongo.
- Example:

```
count = collection.count_documents({})
print(count)
```

This counts all documents in the collection.

Expanded Explanation:

- You can pass a filter object to count_documents({ ... }) to count only matching documents.
- For advanced counting and metrics, consider using the MongoDB aggregation framework.

Final Notes

This presentation covers the fundamentals of document databases, JSON/BSON data formats, and MongoDB. Key takeaways include:

- **Document Databases** store semi-structured data in JSON-like formats, offering flexibility and scalability.
- **MongoDB** is a leading document database, well-suited for modern applications requiring agile schema evolution and horizontal scaling.
- **Core Operations** in MongoDB—like find(), insert(), update(), delete()—are analogous to SQL's CRUD but use JSON-based query syntax.

- **Tools** such as the MongoDB shell (mongosh), MongoDB Compass, and language-specific drivers (e.g., PyMongo) make it straightforward to integrate MongoDB into various development workflows.

By understanding these concepts, you can decide if a document-oriented approach is right for your application’s data needs and start leveraging MongoDB’s powerful feature set.