Chapter2: MongoDB

Title: Introduction to MongoDB
Subtitle: A NoSQL Database for Modern Applications
Presented by: [Ahmad Hamo]
Date: [3-5-2025]

What is MongoDB?

- **Definition:** MongoDB is a popular NoSQL database that stores data in flexible, JSON-like documents.
- **Key Features:**
  - Document-oriented (BSON format).
  - Schema-less (flexible data models).
  - High performance, scalability, and availability.
  - Rich query language and aggregation framework.

MongoDB Structure

- **Document:**
  - Basic unit (like a row in SQL).
  - Stored in BSON (Binary JSON).
  - Example from `sample_mflix.movies`:

```
{
  "_id": ObjectId("573a1390f29313caabcd42e8"),
  "title": "The Great Train Robbery",
  "year": 1903,
  "genres": ["Short", "Western"]
}
```

- **Collection:** Group of documents (e.g., `movies`).
- **Database:** Contains multiple collections (e.g., `sample_mflix`).

MongoDB vs. Relational Databases

Feature	MongoDB (NoSQL)	Relational (SQL)
Data Model	Documents (BSON)	Tables (Rows & Columns)
Schema	Flexible (Dynamic)	Fixed (Predefined)
Query Language	MongoDB Query Language	SQL
Scalability	Horizontal (Sharding)	Vertical

MongoDB Terminologies for RDBMS concepts

RDBMS		MongoDB
Database	⇒	Database
Table, View	⇒	Collection
Row	⇒	Document (JSON, BSON)
Column	⇒	Field
Index	⇒	Index
Join	⇒	Embedded Document
Foreign Key	⇒	Reference
Partition	⇒	Shard

5

Why Use MongoDB?

- **Flexibility:** No rigid schema (e.g., add rating to one movie only).
- **Performance:** Optimized for high-speed queries (e.g., find() with filters).
- **Scalability:** Horizontal scaling via sharding.

- MongoDB stores documents (or) objects.
- Now-a-days, everyone works with objects (Python/Ruby/Java/etc.)
- And we need Databases to persist our objects. Then why not store objects directly ?

- **Use Cases:**
  - Real-time analytics (aggregations).
  - Content management (flexible documents).

NoSQL Definition- *From www.nosql-database.org*

Next Generation Databases mostly addressing some of the points:

- **non-relational,**
- **distributed,**
- **open-source** and
- **horizontal scalable**.

Often more characteristics apply as:

- **schema-free,**
- **easy replication support,**
- **simple API,**
- **eventually consistent** / **BASE** (not ACID),
- **huge data amount**, and more.

NoSQL Distinguishing Characteristics

Large data volumes

- Google's "big data"

Scalable replication and distribution

- Potentially thousands of machines
- Potentially distributed around the world

Queries need to return answers quickly

Schema-less

ACID transaction properties are not needed – BASE

CAP Theorem

Open source development

BASE Transactions

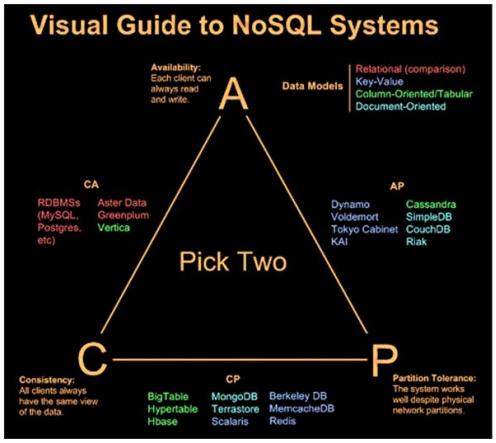
- Acronym contrived to be the opposite of ACID
  - **B**asically **A**vailable,
  - **S**oft state,
  - **E**ventually Consistent
- Characteristics
  - Weak consistency – stale data OK
  - Availability first
  - Best effort
  - Approximate answers OK
  - Aggressive (optimistic)
  - Simpler and faster

CAP Theorem

A congruent and logical way for assessing the problems involved in assuring ACID-like guarantees in distributed systems is provided by the CAP theorem

At most two of the following three can be maximized at one time

- Consistency
  - Each client has the same view of the data
- Availability
  - Each client can always read and write
- Partition tolerance
  - System works well across distributed physical networks



CAP Theorem: Two out of Three

- CAP theorem – At most two properties on three can be addressed
- The choices could be as follows:
  1. **Availability is compromised** but consistency and partition tolerance are preferred over it
  2. The system has little or **no partition tolerance**. Consistency and availability are preferred
  3. **Consistency is compromised** but systems are always available and can work when parts of it are partitioned

MongoDB Demo

```
show dbs;
use demo;
show collections;
db.books.insertOne({title: "MongoDB",
    likes: 100});
db.books.find();
show collections;
show dbs;
db.books.insertMany([{title: "a"},
    {name: "b"}]);
```

```
db.books.find();
db.books.find({likes: 100});
db.books.find({likes: {$gt: 10}});
db.books.updateOne({title: "MongoDB"},
{$set: { likes: 200 }});
db.books.find();
db.books.deleteOne({title: "a"});
db.books.drop();
show collections;
show dbs;
```

Querying Data (Chapter02-Querying-in-MongoDB)

Basic Query

```
// Find movies released in 1969 (limit to 5 results)
db.movies.find({"year": 1969}).limit(5);
```

Filter with Conditions

```
// Find USA comedies released after 1945
db.movies.find({
  "year": { $gt: 1945 },
  "countries": "USA",
  "genres": "Comedy"
});
```

Projection & Sorting (Chapter02-Projection)

```
// Project only title, countries, and year (exclude _id)
db.movies.find(
  { "year": { $gt: 1945 }, "countries": "USA", "genres": "Comedy" },
  { "_id": 0, "title": 1, "countries": 1, "year": 1 }
).sort({ "year": 1 }).limit(5);
```

Output Example:

```
{ "title": "A Comedy Movie", "countries": ["USA"], "year": 1950 }
```

CRUD Operations in MongoDB

Create (Chapter02-Creating-new-Documents)

```
// Insert one document
db.movies.insertOne({
  title: "Once upon a time on Moon",
  genres: ["Test"],
  year: 2024
});

// Insert multiple documents
db.movies.insertMany([
  { title: "Once upon a time on Mars", genres:
["Test"], year: 2023 },
  { title: "Tiger Force in Paradise", genres:
["Test"], year: 2019 }
]);
```

Updating Documents (Chapter02-Updating-Documents)

```
// Update one document
db.movies.updateOne(
  { genres: "Test" },
  { $set: { "genres.$": "PlaceHolder" } }
);

// Update multiple documents (+ increment year)
db.movies.updateMany(
  { "genres": "Test" },
  {
    $set: { "genres.$": "PlaceHolder" },
    $inc: { "year": 1 }
  }
);
```

Deleting Documents (`Chapter02-Deleting-Documents`)

```
// Delete all movies with "PlaceHolder" genre
db.movies.deleteMany({ genres: "PlaceHolder" });
```

Note: Use `deleteOne()` to remove the first matching document.

Aggregation Framework (`Chapter02-Aggregation_Frameworks`)

Simple Aggregation

```
// Match comedy movies
db.movies.aggregate([{ $match: { "genres": "Comedy" } }]);
```

Group & Calculate Averages

```
// Average runtime of comedy movies
db.movies.aggregate([
  { $match: { type: "movie", genres: "Comedy" } },
  { $group: { _id: null, averageRuntime: { $avg: "$runtime"
} } }
]);
```

Output:

```
{ "_id": null, "averageRuntime": 98.86 }
```

MongoDB Aggregates

- MongoDB supports complex queries through “aggregates”
- MongoDB aggregates are very much like SQL SELECT queries
  - **stages** – SQL SELECT clause
  - **pipeline** – SQL SELECT statement

- Aggregation
  - Aggregations are operations that process data records and return computed results.

- What is an Aggregation Pipeline?
  - An aggregation pipeline is a series of document transformations which are executed in stages

MongoDB Aggregates: **Example**

- { _id: 1, cust_id: "a", status: "A", amount: 50 }
- { _id: 2, cust_id: "a", status: "A", amount: 100 }
- { _id: 3, cust_id: "c", status: "D", amount: 25 }
- { _id: 4, cust_id: "d", status: "C", amount: 125 }
- { _id: 5, cust_id: "d", status: "A", amount: 25 }

```
db.orders.aggregate([
 { $match: { status: "A" } },
 { $group: {
     _id: "$cust_id",
     total: { $sum: "$amount" },
     count: { $sum: 1 }
   }
 },
 { $sort: { total: -1 } }
])
```

- Equivalent to SQL SELECT
  - Just $match is fine, for example
  - In $group stage, _id is “group by attributes”

Pipeline stages:

1. **`$match: { status: "A" }`** This filters the documents to only those where `status` is `"A"`:

```
{ _id: 1, cust_id: "a", status: "A", amount: 50 }
{ _id: 2, cust_id: "a", status: "A", amount: 100 }
{ _id: 5, cust_id: "d", status: "A", amount: 25 }
```

2. **`$group` stage** Group by `cust_id`, and compute:
   - `total`: sum of `amount`
   - `count`: count of matching documents per customer

   Grouping:
   - `cust_id: "a"` → total = 50 + 100 = **150**, count = 2
   - `cust_id: "d"` → total = 25, count = 1

   So the result after grouping:

```
{ _id: "a", total: 150, count: 2 }
{ _id: "d", total: 25, count: 1 }
```

3. **`$sort: { total: -1 }`** Sort by `total` descending:

   Final output:

```
{ _id: "a", total: 150, count: 2 }
{ _id: "d", total: 25, count: 1 }
```

Common Aggregate Stages

- $**match** ≈ WHERE
- $**group** ≈ GROUP BY
- $**sort** ≈ ORDER BY
- $**limit** ≈ FETCH FIRST
- $**project** ≈ SELECT
- $**unwind**: replicate document per every element in the array
  - {$unwind: “y” }: {“x”: 1, “y”: [1, 2] } -> {”x”: 1, “y”: 1}, {“x”: 1, “y”: 2 }
- $**lookup**: “look up and join” another document based on the attribute value
  - {$lookup: { from: <collection to join>, localField: <local join attr>, foreignField: <remote join attr>, as: <output field name> }}
  - Matching documents are returned as an array in <output field name>

More on MongoDB aggregates

- Short tutorial: https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/
- Reference: https://docs.mongodb.com/manual/reference/method/db.collection.aggregate/

Basic MongoDB Commands (1)

- **mongo**: start MongoDB shell
- **use <dbName>:** use the database
- **show dbs**: show list of databases
- **show collect**ions: show list of collections
- **db.colName.drop():** delete `colName` collection
- **db.dropDatabase()**: delete current database

Basic MongoDB Commands (2)

- CRUD operations
  - **insertOne(), insertMany()**
  - **findOne(), find()**
  - **updateOne(), updateMany()**
  - **deleteOne(), deleteMany()**
- Insertion: **insertX(doc(s))**
  - db.books.**insertOne**({title: "MongoDB", likes: 100})
  - db.books.**insertMany**([{title: "a"}, {title: "b"}])

Basic MongoDB Commands (3)

- Retrieval: **findX(condition)**
  - db.books.**findOne**({likes: 100})
  - db.books.**find**({$and: [{likes: {$gte: 10}}, {likes: {$lt: 20}}]})
    - Other Boolean/comaprision operators: $or, $not, $gt, $ne, …
- Update: **updateX(condition, update_operation)**
  - db.books.**updateOne**({title: "MongoDB"}, {$set: {title: "MongoDB II"}})
  - db.books.**updateMany**({title: "MongoDB"}, {$inc: {likes: 1}})
    - Other update operators: $mul (multiply), $unset (remove field), …
- Deletion: **deleteX(condition)**
  - db.books.**deleteOne**({title: "MongoDB"})
  - db.books.**deleteMany**({likes: {$lt: 100}})

Summary & Next Steps

- **Key Takeaways:**
  - Documents > Tables, Collections > Schemas.
  - CRUD operations are intuitive (e.g., `insertOne()`, `updateMany()`).
  - Aggregations enable complex analytics.
- **Try It Yourself:**
  - Use MongoDB Atlas (Free Tier).
  - Explore the `sample_mflix` dataset.
- **Demo Suggestions:**
  - Run queries live in `mongosh` or Compass.
  - Show the `sample_mflix` dataset (in Atlas).
- **Emphasize:**
  - Schema flexibility (e.g., adding `rating` to one document).
  - Power of aggregations (e.g., `$group` for analytics).