Last Recitation: MongoDB Overview

Table of Contents i

Introduction to MongoDB

What is MongoDB? i

- Document-oriented NoSQL database
- Stores data in flexible, JSON-like BSON documents
- Designed for scalability, performance, and high availability
- Released in 2009, now one of the most popular databases
- Written in C++
- Open-source with commercial support options

Key MongoDB Features i

- Document-based storage (BSON format)
- Dynamic schema support
- Powerful query language
- Horizontal scaling via sharding
- High availability with replica sets
- Geospatial indexing
- Aggregation framework
- Full-text search
- Native time series support
- Transaction support (since v4.0)

MongoDB vs. Traditional Relational Databases i

MongoDB

- Documents (BSON)
- Collections
- Dynamic schema
- Embedded documents
- Horizontal scaling

Relational DB

- Records (Rows)
- Tables
- Fixed schema
- Foreign keys/Joins
- Vertical scaling

Setting Up MongoDB

Installing MongoDB i

On Windows:

- Download installer from mongodb.com
- Run the installer and follow wizard
- Add MongoDB bin directory to PATH

On MacOS:

```
brew tap mongodb/brew
brew install mongodb-community
```

On Ubuntu:

```
sudo apt update
sudo apt install -y mongodb
```

Installing MongoDB ii

Docker:

```
1 docker pull mongo
2 docker run --name mongodb -d -p 27017:27017 mongo
```

MongoDB Shell and GUI Tools i

MongoDB Shell (mongosh):

```
mongosh
```

GUI Tools:

- MongoDB Compass - Official GUI client
- Studio 3T - Formerly MongoChef
- NoSQLBooster - Advanced query builder
- Robo 3T - Lightweight option

Connect string format:

```
mongodb://[username:password@]host[:port]/[database]
```

Basic MongoDB Concepts

MongoDB Document Structure

Documents are stored in BSON format (Binary JSON)

```
{
   _id: ObjectId("507f1f77bcf86cd799439011"),
   name: "John Doe",
   age: 30,
   email: "john@example.com",
   address: {
      street: "123 Main St",
      city: "New York",
      state: "NY",
      zip: "10001"
   },
   hobbies: ["reading", "hiking", "photography"],
   active: true,
   created: ISODate("2022-04-15T10:30:00Z")
}
```

MongoDB Data Types i

- **String**: UTF-8 character strings
- **Integer**: 32-bit or 64-bit integers
- **Double**: 64-bit floating point values
- **Boolean**: true or false
- **Date**: milliseconds since Unix epoch as 64-bit integer
- **Object**: Embedded documents
- **Array**: Lists of values
- **ObjectId**: 12-byte identifier (default for _id)
- **Null**: Null value
- **Binary data**: Binary data (images, files)
- **Regular expression**: Pattern matching
- **JavaScript code**: Stored JavaScript functions

Database and Collection Operations i

Show and Create Databases:

```
// Show existing databases
show dbs

// Create/switch to a database
use myDatabase

// Show current database
db
```

Collection Operations:

Database and Collection Operations ii

```
// Show collections in current database
show collections

// Create collection explicitly
db.createCollection("users")

// Drop (delete) a collection
db.users.drop()

// Rename collection
db.users.renameCollection("customers")
```

CRUD Operations

Creating Documents (Insert) i

Insert a single document:

```
db.users.insertOne({
  name: "Alice Smith",
  email: "alice@example.com",
  age: 32,
  active: true
})
```

Insert multiple documents:

Creating Documents (Insert) ii

```
db.users.insertMany([
  {
    name: "Bob Johnson",
    email: "bob@example.com",
    age: 28,
    active: true
  },
  {
    name: "Carol White",
    email: "carol@example.com",
    age: 41,
    active: false
  }
])
```

Reading Documents (Query) i

Find all documents in a collection:

```
db.users.find()
```

Find with conditions:

```
// Find users over 30
db.users.find({ age: { $gt: 30 } })

// Find active users
db.users.find({ active: true })

// Find specific user by name
db.users.find({ name: "Alice Smith" })
```

Format results:

Reading Documents (Query) ii

```
// Pretty print results
db.users.find().pretty()

// Limit results
db.users.find().limit(5)
```

Querying Embedded Documents i

Find by embedded field:

```
// Find users living in New York
db.users.find({ "address.city": "New York" })

// Find users with a specific hobby
db.users.find({ hobbies: "hiking" })
```

Return only specific fields:

```
// Return only name and email
db.users.find(
  { age: { $gt: 30 } },
  { name: 1, email: 1, _id: 0 }
)
```

Count documents:

Querying Embedded Documents ii

```
db.users.countDocuments({ active: true })
```

Query Operators i

Comparison Operators:

```
// Greater than
db.users.find({ age: { $gt: 30 } })

// Less than or equal to
db.users.find({ age: { $lte: 25 } })

// Equal to
db.users.find({ age: { $eq: 32 } })

// Not equal to
db.users.find({ age: { $ne: 32 } })

// In array of values
db.users.find({ age: { $in: [25, 30, 35] } })

```

Query Operators ii

```
16 // Not in array
17 db.users.find({ age: { $nin: [25, 30, 35] } })
```

Logical Operators i

AND, OR, NOT:

```
// AND - both conditions must be true
db.users.find({
  $and: [
    { age: { $gt: 30 } },
    { active: true }
  ]
})

// OR - either condition can be true
db.users.find({
  $or: [
    { age: { $lt: 25 } },
    { age: { $gt: 50 } }
  ]
})
```

Logical Operators ii

```
16
17 // NOT - negates the condition
18 db.users.find({
19   age: { $not: { $lt: 30 } }
20 })
```

Element Operators i

Field existence and type:

```
// Field exists
db.users.find({ email: { $exists: true } })

// Field does not exist
db.users.find({ phone: { $exists: false } })

// Field is of specific type
db.users.find({ age: { $type: "number" } })
db.users.find({ name: { $type: "string" } })
```

Regular Expression:

Element Operators ii

```
// Name starts with 'A'
db.users.find({ name: /^A/ })

// Case insensitive email search
db.users.find({ email: /gmail\.com$/i })
```

Array Operators i

Array operations:

```
// Array contains element
db.users.find({ hobbies: "reading" })

// Array contains all elements
db.users.find({ hobbies: { $all: ["reading", "hiking"] }
    })

// Array size
db.users.find({ hobbies: { $size: 3 } })

// Array element matches conditions
db.users.find({
  "scores": { $elemMatch: { $gt: 80, $lt: 90 } }
})
```

Updating Documents i

Update a single document:

```
// Replace entire document
db.users.replaceOne(
  { name: "Alice Smith" },
  {
    name: "Alice Johnson",
    email: "alice.j@example.com",
    age: 33,
    active: true
  }
)

// Update specific fields
db.users.updateOne(
  { name: "Bob Johnson" },
  { $set: { age: 29, email: "bob.new@example.com" } }
```

Updating Documents ii

```
16 )
```

Updating Multiple Documents i

Update many documents:

```
// Increment age by 1 for all users
db.users.updateMany(
  {},
  { $inc: { age: 1 } }
)

// Set active status for users over 40
db.users.updateMany(
  { age: { $gt: 40 } },
  { $set: { active: false } }
)
```

Upsert (insert if not exists):

Updating Multiple Documents ii

```
db.users.updateOne(
  { email: "david@example.com" },
  { $set: { name: "David Brown", age: 35, active: true }
     },
  { upsert: true }
)
```

Update Operators i

Field update operators:

```
// Set field values
db.users.updateOne(
  { _id: ObjectId("507f1f77bcf86cd799439011") },
  { $set: { status: "Premium" } }
)

// Increment numeric fields
db.users.updateOne(
  { name: "Alice Johnson" },
  { $inc: { loginCount: 1, age: 1 } }
)

// Multiply numeric fields
db.users.updateOne(
  { name: "Bob Johnson" },
```

Update Operators ii

```
  { $mul: { score: 1.1 } } // Increase score by 10%
)

// Remove fields
db.users.updateOne(
  { name: "Carol White" },
  { $unset: { temporary: "", oldField: "" } }
)
```

Array Update Operators i

Array modifications:

```
// Add to array if not exists
db.users.updateOne(
  { name: "Alice Johnson" },
  { $addToSet: { hobbies: "cooking" } }
)

// Push to array (can add duplicates)
db.users.updateOne(
  { name: "Bob Johnson" },
  { $push: { hobbies: "swimming" } }
)

// Push multiple items
db.users.updateOne(
  { name: "Carol White" },
```

Array Update Operators ii

```
  { $push: {
      hobbies: {
        $each: ["swimming", "yoga", "painting"]
      }
    }
  }
)

// Remove from array
db.users.updateOne(
  { name: "Bob Johnson" },
  { $pull: { hobbies: "hiking" } }
)
```

Deleting Documents i

Delete a single document:

```
db.users.deleteOne({ name: "Alice Johnson" })
```

Delete multiple documents:

```
// Delete inactive users
db.users.deleteMany({ active: false })

// Delete users older than 60
db.users.deleteMany({ age: { $gt: 60 } })
```

Delete all documents:

```
db.users.deleteMany({})
```

Drop entire collection:

Deleting Documents ii

```
1 db.users.drop()
```

Indexes and Performance

Creating Indexes i

Create a single field index:

```
// Create ascending index on email
db.users.createIndex({ email: 1 })

// Create descending index on age
db.users.createIndex({ age: -1 })
```

Create compound index:

```
// Index on both age and name
db.users.createIndex({ age: 1, name: 1 })
```

Create unique index:

Creating Indexes ii

```
// Ensure email uniqueness
db.users.createIndex(
  { email: 1 },
  { unique: true }
)
```

Managing Indexes i

List all indexes:

```
db.users.getIndexes()
```

Drop an index:

```
// Drop by name
db.users.dropIndex("email_1")

// Drop by key specification
db.users.dropIndex({ email: 1 })
```

Special index types:

Managing Indexes ii

```
// Create text index for full-text search
db.products.createIndex({ description: "text" })

// Create geospatial index
db.places.createIndex({ location: "2dsphere" })

// Create TTL index (automatic document expiration)
db.sessions.createIndex(
  { lastUpdated: 1 },
  { expireAfterSeconds: 3600 } // 1 hour
)
```

Query Performance i

Explain execution plan:

```
db.users.find({ age: { $gt: 30 } }).explain()

// Execution stats
db.users.find({ age: { $gt: 30 } })
  .explain("executionStats")
```

Using covered queries:

```
// Query only uses indexed fields
db.users.createIndex({ age: 1, name: 1 })

// Covered query (only returns indexed fields)
db.users.find(
  { age: { $gt: 30 } },
  { age: 1, name: 1, _id: 0 }
```

Query Performance ii

```
8 )
```

Advanced Queries

Sorting and Pagination i

Sort results:

```
// Sort by age ascending
db.users.find().sort({ age: 1 })

// Sort by multiple fields
db.users.find().sort({ active: -1, age: 1 })
```

Pagination:

```
// Skip first 20, return next 10
db.users.find()
  .sort({ name: 1 })
  .skip(20)
  .limit(10)
```

Count with conditions:

Sorting and Pagination ii

```
db.users.countDocuments({ age: { $gt: 30 } })
```

Aggregation Framework i

What is Aggregation?

- Processing data records and returning computed results
- Pipeline of transformations on documents
- More powerful than simple find/filter
- Like SQL's GROUP BY, JOIN, etc.

Simple aggregation example:

Aggregation Framework ii

```
db.users.aggregate([
  { $match: { active: true } },
  { $group: {
      _id: null,
      avgAge: { $avg: "$age" },
      count: { $sum: 1 }
    }
  }
])
```

Aggregation Pipeline Stages i

Common stages in the aggregation pipeline:

```
db.orders.aggregate([
  // Filter documents
  { $match: { status: "completed" } },

  // Group and calculate
  { $group: {
      _id: "$customerId",
      totalSpent: { $sum: "$total" },
      orderCount: { $sum: 1 }
    }
  },

  // Sort by result
  { $sort: { totalSpent: -1 } },

```

Aggregation Pipeline Stages ii

```
16    // Limit results
17    { $limit: 5 },
18
19    // Reshape output document
20    { $project: {
21        customerId: "$_id",
22        _id: 0,
23        totalSpent: 1,
24        orderCount: 1,
25        avgOrderValue: { $divide: ["$totalSpent", "
      $orderCount"] }
26      }
27    }
28 ])
```

Aggregation Examples i

Group by field and calculate statistics:

```
db.sales.aggregate([
  { $match: { date: {
      $gte: ISODate("2023-01-01"),
      $lt: ISODate("2024-01-01")
    }}
  },
  { $group: {
      _id: "$category",
      totalSales: { $sum: "$amount" },
      count: { $sum: 1 },
      avgSale: { $avg: "$amount" },
      minSale: { $min: "$amount" },
      maxSale: { $max: "$amount" }
    }
  },
```

Aggregation Examples ii

```
16   { $sort: { totalSales: -1 } }
17 ])
```

More Aggregation Operations i

Unwind arrays and group:

```
// Find popular tags
db.products.aggregate([
  // Unwind tag array into separate documents
  { $unwind: "$tags" },

  // Group by tag name and count
  { $group: {
      _id: "$tags",
      count: { $sum: 1 }
    }
  },

  // Sort by popularity
  { $sort: { count: -1 } },

```

More Aggregation Operations ii

```
16   // Take top 10
17   { $limit: 10 }
18 ])
```

Lookups (Joins) in Aggregation i

Joining data from multiple collections:

```
db.orders.aggregate([
  { $match: {
      status: "shipped",
      orderDate: { $gte: ISODate("2023-01-01") }
    }
  },
  // Join with customers collection
  { $lookup: {
      from: "customers",
      localField: "customerId",
      foreignField: "_id",
      as: "customerInfo"
    }
  },
  // Unwind the resulting array
```

Lookups (Joins) in Aggregation ii

```
16    { $unwind: "$customerInfo" },
17    // Shape the output
18    { $project: {
19        orderNumber: 1,
20        orderDate: 1,
21        total: 1,
22        "customer.name": "$customerInfo.name",
23        "customer.email": "$customerInfo.email"
24      }
25    }
26 ])
```

MongoDB Security

Authentication and Authorization i

Create a user:

```
use admin
db.createUser({
  user: "adminUser",
  pwd: "securePassword",
  roles: [{ role: "userAdminAnyDatabase", db: "admin" }]
})
```

Create application user:

Authentication and Authorization ii

```
use myDatabase
db.createUser({
  user: "appUser",
  pwd: "appPassword",
  roles: [
    { role: "readWrite", db: "myDatabase" },
    { role: "read", db: "analytics" }
  ]
})
```

Authenticate:

```
mongosh -u appUser -p appPassword --
    authenticationDatabase myDatabase
```

Built-in Roles i

Database User Roles:

- `read`: Read-only access
- `readWrite`: Read and write access

Database Administration Roles:

- `dbAdmin`: Administrative tasks without data access
- `dbOwner`: All operations on a database
- `userAdmin`: Manage users and roles

Cluster Administration Roles:

- `clusterAdmin`: Highest cluster management
- `clusterManager`: Management and monitoring

Built-in Roles ii

- `clusterMonitor`: Monitoring only

Backup and Restore Roles:

- backup: Backup data
- `restore`: Restore data

MongoDB Transactions

Multi-Document Transactions i

Since MongoDB 4.0, transactions across multiple documents and collections:

```
// Start a session
const session = db.getMongo().startSession();
session.startTransaction();

try {
  const accounts = session.getDatabase("banking").
    accounts;

  // Withdraw from account A
  accounts.updateOne(
    { _id: "A" },
    { $inc: { balance: -100 } }
  );
```

Multi-Document Transactions ii

```
13
14    // Deposit to account B
15    accounts.updateOne(
16      { _id: "B" },
17      { $inc: { balance: 100 } }
18    );
19
20    // Commit the transaction
21    session.commitTransaction();
22  } catch (error) {
23    // Abort on error
24    session.abortTransaction();
25    print("Transaction aborted:", error);
26  } finally {
27    session.endSession();
28  }
```

Practical Examples

Example: E-commerce Product Catalog i

Product schema design:

```
// Products collection
{
  _id: ObjectId("60a2e8f35a6fbd1a58ee0f20"),
  sku: "P12345",
  name: "Wireless Headphones",
  brand: "AudioMax",
  category: "Electronics",
  price: 89.99,
  discount: 0,
  inStock: true,
  details: {
    color: "Black",
    weight: "250g",
    dimensions: "7.5 x 6.1 x 3.2 inches",
    connectivity: "Bluetooth 5.0",
```

Example: E-commerce Product Catalog ii

```
    batteryLife: "30 hours"
  },
  variants: [
    { color: "Black", sku: "P12345-BLK", inventory: 253 },
    { color: "White", sku: "P12345-WHT", inventory: 167 },
    { color: "Blue", sku: "P12345-BLU", inventory: 92 }
  ],
  tags: ["wireless", "headphones", "bluetooth", "audio"],
  ratings: {
    average: 4.5,
    count: 137
  },
  created: ISODate("2023-04-10"),
```

Example: E-commerce Product Catalog iii

```
  updated: ISODate("2023-05-01")
}
```

Query examples:

```
// Find products by category with price range
db.products.find({
  category: "Electronics",
  price: { $gte: 50, $lte: 100 }
})

// Find products with specific tag and in stock
db.products.find({
  tags: "bluetooth",
  inStock: true
})

```

Example: E-commerce Product Catalog iv

```
// Find products with specific variant color available
db.products.find({
  "variants.color": "White",
  "variants.inventory": { $gt: 0 }
})
```

Example: User Analytics i

Track user sessions:

```
// User sessions collection
{
  _id: ObjectId("60a2e8f35a6fbd1a58ee0f30"),
  userId: ObjectId("507f1f77bcf86cd799439011"),
  sessionStart: ISODate("2023-05-15T10:23:45Z"),
  sessionEnd: ISODate("2023-05-15T11:05:12Z"),
  device: {
    type: "mobile",
    os: "iOS",
    browser: "Safari",
    screenSize: "390x844"
  },
  location: {
    country: "US",
    city: "Boston",
```

Example: User Analytics ii

```
    ip: "192.168.1.1" // anonymized for example
  },
  actions: [
    {
      type: "pageView",
      page: "/products",
      timestamp: ISODate("2023-05-15T10:24:12Z")
    },
    {
      type: "productView",
      productId: ObjectId("60a2e8f35a6fbd1a58ee0f20"),
      timestamp: ISODate("2023-05-15T10:26:33Z")
    },
    {
      type: "addToCart",
      productId: ObjectId("60a2e8f35a6fbd1a58ee0f20"),
```

Example: User Analytics iii

```
32        quantity: 1,
33        timestamp: ISODate("2023-05-15T10:28:15Z")
34      }
35    ]
36  }
```

Example: Analytics Aggregation i

Popular products by views:

```
db.sessions.aggregate([
  // Unwind the actions array
  { $unwind: "$actions" },

  // Filter for productView actions
  { $match: { "actions.type": "productView" } },

  // Group by product and count views
  { $group: {
      _id: "$actions.productId",
      viewCount: { $sum: 1 }
    }
  },

  // Sort by most viewed
```

Example: Analytics Aggregation ii

```
16  { $sort: { viewCount: -1 } },
17
18  // Take top 10
19  { $limit: 10 },
20
21  // Join with products collection
22  { $lookup: {
23      from: "products",
24      localField: "_id",
25      foreignField: "_id",
26      as: "product"
27    }
28  },
29
30  // Unwrap product array
31  { $unwind: "$product" },
```

Example: Analytics Aggregation iii

```
32
33    // Shape output
34    { $project: {
35        _id: 0,
36        productId: "$_id",
37        name: "$product.name",
38        category: "$product.category",
39        viewCount: 1
40      }
41    }
42 ])
```

Example: Geospatial Queries i

Store with geospatial data:

```
// Stores collection
db.stores.insertMany([
  {
    name: "Downtown Store",
    location: {
      type: "Point",
      coordinates: [-73.9857, 40.7484] // longitude, latitude
    },
    address: "123 Main St, New York, NY"
  },
  {
    name: "Uptown Store",
    location: {
      type: "Point",
```

Example: Geospatial Queries ii

```
      coordinates: [-73.9654, 40.7829]
    },
    address: "456 Park Ave, New York, NY"
  }
])

// Create geospatial index
db.stores.createIndex({ location: "2dsphere" })
```

Example: Finding Nearby Locations i

Find stores near a point:

```
// Find stores within 5km of a location
db.stores.find({
  location: {
    $near: {
      $geometry: {
        type: "Point",
        coordinates: [-73.9759, 40.7628] // Times Square
      },
      $maxDistance: 5000 // 5km in meters
    }
  }
})

// Find stores within a specific area (polygon)
db.stores.find({
```

Example: Finding Nearby Locations ii

```
  location: {
    $geoWithin: {
      $geometry: {
        type: "Polygon",
        coordinates: [[
          [-74.0107, 40.7128], // Lower Manhattan
          [-73.9654, 40.7128],
          [-73.9654, 40.7484],
          [-74.0107, 40.7484],
          [-74.0107, 40.7128]
        ]]
      }
    }
  }
})
```

MongoDB in Python

Using MongoDB with Python:

```
# Install driver: pip install pymongo

from pymongo import MongoClient
from datetime import datetime

# Connect to MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client['myDatabase']
users = db['users']

# Insert a document
result = users.insert_one({
    'name': 'John Doe',
    'email': 'john@example.com',
    'age': 30,
```

MongoDB with Python ii

```
    'created': datetime.now()
})
print(f"Inserted document with ID: {result.inserted_id}")

# Query documents
for user in users.find({'age': {'$gt': 25}}):
    print(f"Found user: {user['name']}, age: {user['age']}")

# Update a document
update_result = users.update_one(
    {'name': 'John Doe'},
    {'$set': {'age': 31}}
)
```

MongoDB with Python iii

```
29 print(f"Modified {update_result.modified_count} document
       (s)")
30
31 # Delete a document
32 delete_result = users.delete_one({'name': 'John Doe'})
33 print(f"Deleted {delete_result.deleted_count} document(s
       )")
```

MongoDB Atlas and Cloud

MongoDB Atlas i

What is MongoDB Atlas?

- Fully managed MongoDB cloud service
- Available on AWS, Azure, and Google Cloud
- Auto-scaling and backups
- Security features (encryption, VPC peering, IP whitelisting)
- Built-in monitoring and alerting
- Free tier available for learning

Connection string format:

```
mongodb+srv://<username>:<password>@<cluster-url>/<
    database>
```

Features:

MongoDB Atlas ii

- Online Archive
- Data Lake (Atlas Data Lake)
- Full-Text Search
- Charts for data visualization
- Realm (mobile and edge solutions)

MongoDB Compass i

Key features of MongoDB Compass:

- Visual exploration of your data
- CRUD operations via UI
- Query builder
- Index management
- Performance analysis
- Aggregation pipeline builder
- Schema visualization

Download from: mongodb.com/products/compass

Conclusion

Resources to Learn More i

- Official Documentation: docs.mongodb.com
- MongoDB University: university.mongodb.com (free courses)
- MongoDB Developer Hub: mongodb.com/developer
- GitHub: github.com/mongodb/mongo
- Stack Overflow: stackoverflow.com/questions/tagged/mongodb
- MongoDB Blog: mongodb.com/blog

Thank You!

Questions?

Thank you for your attention!