04 basics & SuperHeroes DB - KirkYagami

MongoDB Basics

Introduction to MongoDB

MongoDB is a NoSQL, document-oriented database that stores data in flexible, JSON-like documents called BSON (Binary JSON). Unlike traditional relational databases, MongoDB doesn't require a predefined schema, making it highly flexible for modern applications.

Key Concepts:

- **Document**: A record in MongoDB, similar to a row in SQL
- **Collection**: A group of documents, similar to a table in SQL
- **Database**: A container for collections
- **Field**: A key-value pair in a document, similar to a column in SQL

Databases and Collections

Creating and Using Databases

```
// Switch to or create a database
use superheroDB

// Check current database
db

// List all databases
show dbs

// Drop a database
db.dropDatabase()
```

Working with Collections

```
// Collections are created automatically when you insert documents
// List all collections in current database
show collections

// Create a collection explicitly
db.createCollection("heroes")

// Drop a collection
db.heroes.drop()
```

Data Types

MongoDB supports various data types. Here are the most common ones:

Common Data Types:

- **String**: Text data
- **Number**: Integer or floating-point numbers
- **Boolean**: true/false values

- **Date**: Date and time values
- **Array**: List of values
- **Object**: Embedded documents
- **ObjectId**: Unique identifier for documents
- **Null**: Represents null values

Example Document Structure:

```
{
  _id: ObjectId("..."),
  name: "Spider-Man",
  realName: "Peter Parker",
  age: 23,
  isActive: true,
  powers: ["web-slinging", "spider-sense", "wall-crawling"],
  stats: {
    strength: 85,
    speed: 90,
    intelligence: 95
  },
  firstAppearance: new Date("1962-08-01"),
  team: null
}
```

Insert Operations

Insert Single Document

```
// Insert one superhero
db.heroes.insertOne({
  name: "Spider-Man",
  realName: "Peter Parker",
  age: 23,
  isActive: true,
  powers: ["web-slinging", "spider-sense", "wall-crawling"],
  stats: {
    strength: 85,
    speed: 90,
    intelligence: 95
  },
  firstAppearance: new Date("1962-08-01"),
  city: "New York"
})
```

Insert Multiple Documents

```
// Insert multiple superheroes
db.heroes.insertMany([
  {
    name: "Batman",
    realName: "Bruce Wayne",
    age: 35,
    isActive: true,
    powers: ["martial arts", "detective skills", "gadgets"],
```

```
    stats: {
      strength: 75,
      speed: 80,
      intelligence: 100
    },
    firstAppearance: new Date("1939-05-01"),
    city: "Gotham"
  },
  {
    name: "Wonder Woman",
    realName: "Diana Prince",
    age: 3000,
    isActive: true,
    powers: ["super strength", "flight", "lasso of truth"],
    stats: {
      strength: 95,
      speed: 85,
      intelligence: 90
    },
    firstAppearance: new Date("1941-12-01"),
    city: "Washington DC"
  },
  {
    name: "Superman",
    realName: "Clark Kent",
    age: 30,
    isActive: true,
    powers: ["super strength", "flight", "x-ray vision", "heat vision"],
    stats: {
      strength: 100,
      speed: 100,
      intelligence: 85
    },
    firstAppearance: new Date("1938-06-01"),
    city: "Metropolis"
  },
  {
    name: "Iron Man",
    realName: "Tony Stark",
    age: 45,
    isActive: false,
    powers: ["powered armor", "genius intellect", "arc reactor"],
    stats: {
      strength: 70,
      speed: 75,
      intelligence: 100
    },
    firstAppearance: new Date("1963-03-01"),
    city: "New York"
  }
])
```

Find Operations

Basic Find Operations

```
// Find all documents
db.heroes.find()

// Find all documents with pretty formatting
db.heroes.find().pretty()

// Find one document
db.heroes.findOne()

// Find by specific field
db.heroes.find({name: "Spider-Man"})

// Find by multiple fields
db.heroes.find({city: "New York", isActive: true})
```

Projection (Selecting Specific Fields)

```
// Include only specific fields (1 = include, 0 = exclude)
db.heroes.find({}, {name: 1, realName: 1})

// Exclude specific fields
db.heroes.find({}, {_id: 0, powers: 0})

// Find with conditions and projection
db.heroes.find({isActive: true}, {name: 1, city: 1, _id: 0})
```

Querying Nested Documents

```
// Query nested fields using dot notation
db.heroes.find({"stats.strength": 100})

// Query multiple nested fields
db.heroes.find({"stats.strength": {$gte: 90}, "stats.intelligence": {$gte: 90}})
```

Querying Arrays

```
// Find documents where array contains a specific value
db.heroes.find({powers: "flight"})

// Find documents where array contains all specified values
db.heroes.find({powers: {$all: ["super strength", "flight"]}})

// Find by array length
db.heroes.find({powers: {$size: 3}})
```

Comparison Operators

Basic Comparison Operators

```
// Equal to ($eq)
db.heroes.find({age: {$eq: 30}})
// Same as: db.heroes.find({age: 30})
```

```
// Not equal to ($ne)
db.heroes.find({age: {$ne: 30}})

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

// Greater than or equal to ($gte)
db.heroes.find({"stats.strength": {$gte: 90}})

// Less than ($lt)
db.heroes.find({age: {$lt: 35}})

// Less than or equal to ($lte)
db.heroes.find({age: {$lte: 30}})

// In array ($in)
db.heroes.find({city: {$in: ["New York", "Gotham"]}})

// Not in array ($nin)
db.heroes.find({city: {$nin: ["New York", "Gotham"]}})
```

Pattern Matching

```
// Regular expressions
db.heroes.find({name: {$regex: "Man$"}}) // Names ending with "Man"

// Case-insensitive search
db.heroes.find({name: {$regex: "spider", $options: "i"}})

// Exists operator
db.heroes.find({team: {$exists: false}})

// Type operator
db.heroes.find({age: {$type: "number"}})
```

Logical Operators

AND, OR, NOT Operations

```
// AND operator ($and) - all conditions must be true
db.heroes.find({
  $and: [
    {isActive: true},
    {"stats.strength": {$gte: 85}}
  ]
})

// OR operator ($or) - at least one condition must be true
db.heroes.find({
  $or: [
    {city: "New York"},
    {city: "Gotham"}
  ]
})

// NOT operator ($not)
```

```
db.heroes.find({age: {$not: {$gt: 35}}})

// NOR operator ($nor) - none of the conditions should be true
db.heroes.find({
  $nor: [
    {isActive: false},
    {age: {$lt: 25}}
  ]
})
```

Complex Logical Queries

```
// Combining multiple logical operators
db.heroes.find({
  $and: [
    {
      $or: [
        {city: "New York"},
        {city: "Metropolis"}
      ]
    },
    {isActive: true},
    {"stats.intelligence": {$gte: 85}}
  ]
})
```

Sorting and Limiting

Sorting Results

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

// Sort by age (descending)
db.heroes.find().sort({age: -1})

// Sort by multiple fields
db.heroes.find().sort({city: 1, age: -1})

// Sort by nested fields
db.heroes.find().sort({"stats.strength": -1})
```

Limiting Results

```
// Limit to 3 results
db.heroes.find().limit(3)

// Skip first 2 results
db.heroes.find().skip(2)

// Combine skip and limit (pagination)
db.heroes.find().skip(2).limit(2)
```

```
// Combine find, sort, and limit
db.heroes.find({isActive: true}).sort({"stats.strength": -1}).limit(3)
```

Counting Documents

```
// Count all documents
db.heroes.countDocuments()

// Count with conditions
db.heroes.countDocuments({isActive: true})

// Estimated count (faster for large collections)
db.heroes.estimatedDocumentCount()
```

Update Operations

Update Single Document

```
// Update one document using $set
db.heroes.updateOne(
  {name: "Spider-Man"},
  {$set: {age: 24, city: "Queens"}}
)

// Add new field
db.heroes.updateOne(
  {name: "Batman"},
  {$set: {mentor: "Alfred Pennyworth"}}
)

// Update nested fields
db.heroes.updateOne(
  {name: "Iron Man"},
  {$set: {"stats.strength": 80}}
)
```

Update Multiple Documents

```
// Update all documents matching criteria
db.heroes.updateMany(
  {city: "New York"},
  {$set: {state: "NY"}}
)

// Update all documents
db.heroes.updateMany(
  {},
  {$set: {universe: "DC/Marvel"}}
)
```

Array Update Operators

```
// Add element to array ($push)
db.heroes.updateOne(
```

```
  {name: "Spider-Man"},
  {$push: {powers: "enhanced agility"}}
)

// Add multiple elements to array ($push with $each)
db.heroes.updateOne(
  {name: "Batman"},
  {$push: {powers: {$each: ["stealth", "intimidation"]}}}
)

// Remove element from array ($pull)
db.heroes.updateOne(
  {name: "Spider-Man"},
  {$pull: {powers: "enhanced agility"}}
)

// Add to array only if not exists ($addToSet)
db.heroes.updateOne(
  {name: "Superman"},
  {$addToSet: {powers: "invulnerability"}}
)
```

Increment/Decrement Operations

```
// Increment a number field ($inc)
db.heroes.updateOne(
  {name: "Spider-Man"},
  {$inc: {age: 1}}
)

// Increment nested field
db.heroes.updateOne(
  {name: "Batman"},
  {$inc: {"stats.intelligence": 5}}
)
```

Upsert Operations

```
// Update if exists, insert if doesn't exist
db.heroes.updateOne(
  {name: "The Flash"},
  {
    $set: {
      realName: "Barry Allen",
      age: 28,
      powers: ["super speed", "time travel"],
      city: "Central City"
    }
  },
  {upsert: true}
)
```

Delete Operations

Delete Single Document

```
// Delete one document
db.heroes.deleteOne({name: "Iron Man"})

// Delete first matching document
db.heroes.deleteOne({isActive: false})
```

Delete Multiple Documents

```
// Delete all inactive heroes
db.heroes.deleteMany({isActive: false})

// Delete heroes from specific city
db.heroes.deleteMany({city: "Gotham"})

// Delete all documents (careful!)
db.heroes.deleteMany({})
```

Find and Delete

```
// Find and delete in one operation
db.heroes.findOneAndDelete({name: "Superman"})

// Find, delete, and return the deleted document
var deletedHero = db.heroes.findOneAndDelete(
  {age: {$gt: 100}},
  {sort: {age: -1}}
)
```

Indexes

Indexes improve query performance by creating efficient access paths to data.

Creating Indexes

```
// Create single field index
db.heroes.createIndex({name: 1})

// Create compound index
db.heroes.createIndex({city: 1, isActive: 1})

// Create index on nested field
db.heroes.createIndex({"stats.strength": -1})

// Create text index for text search
db.heroes.createIndex({name: "text", realName: "text"})

// Create unique index
db.heroes.createIndex({name: 1}, {unique: true})
```

Managing Indexes

```
// List all indexes
db.heroes.getIndexes()
```

```
// Get index statistics
db.heroes.getIndexes().forEach(printjson)

// Drop an index
db.heroes.dropIndex({name: 1})

// Drop all indexes except _id
db.heroes.dropIndexes()
```

Query Performance

```
// Explain query execution
db.heroes.find({name: "Spider-Man"}).explain("executionStats")

// Use hint to force specific index
db.heroes.find({city: "New York"}).hint({city: 1})
```

Advanced Operations

Aggregation Pipeline

```
// Group heroes by city and count
db.heroes.aggregate([
  {$group: {_id: "$city", count: {$sum: 1}}}
])

// Calculate average strength by city
db.heroes.aggregate([
  {$group: {
    _id: "$city",
    avgStrength: {$avg: "$stats.strength"},
    maxStrength: {$max: "$stats.strength"}
  }}
])

// Filter and project in pipeline
db.heroes.aggregate([
  {$match: {isActive: true}},
  {$project: {name: 1, city: 1, totalPower: {$add: ["$stats.strength", "$stats.speed"]}}},
  {$sort: {totalPower: -1}}
])
```

Text Search

```
// Create text index first
db.heroes.createIndex({name: "text", realName: "text", powers: "text"})

// Perform text search
db.heroes.find({$text: {$search: "spider"}})

// Text search with score
db.heroes.find(
  {$text: {$search: "spider man"}},
```

```
  {score: {$meta: "textScore"}}
).sort({score: {$meta: "textScore"}})
```

Backup and Restore

```
// Export collection to JSON
mongoexport --db superheroDB --collection heroes --out heroes.json

// Import collection from JSON
mongoimport --db superheroDB --collection heroes --file heroes.json

// Create database dump
mongodump --db superheroDB

// Restore database
mongorestore dump/
```

Best Practices

1. Schema Design

- Design your schema based on your query patterns
- Embed related data that's frequently accessed together
- Use references for data that changes frequently or is shared

2. Indexing Strategy

- Index fields that are frequently queried
- Create compound indexes for multi-field queries
- Monitor index usage and remove unused indexes

3. Query Optimization

- Use projections to limit returned data
- Use appropriate operators for range queries
- Limit results when possible

4. Data Validation

```
// Add validation rules
db.createCollection("heroes", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["name", "realName", "age"],
      properties: {
        name: {
          bsonType: "string",
          description: "must be a string and is required"
        },
        age: {
          bsonType: "int",
          minimum: 0,
          maximum: 10000,
          description: "must be an integer between 0 and 10000"
        }
      }
```

```
    }
  }
})
```

Summary

This comprehensive guide covered:

- Database and collection management
- CRUD operations (Create, Read, Update, Delete)
- Query operators (comparison and logical)
- Sorting, limiting, and pagination
- Indexing for performance
- Advanced features like aggregation and text search

Key Takeaways:

1. MongoDB is flexible and schema-less
2. Documents are stored in BSON format
3. Indexes are crucial for performance
4. Aggregation pipeline provides powerful data processing
5. Always consider your query patterns when designing schemas

Next Steps:

- Practice with real datasets
- Learn about sharding and replication
- Explore MongoDB Atlas (cloud service)
- Study advanced aggregation operations
- Learn about transactions in MongoDB