**Course Code:** CSE 3513

**Course Name:** NoSQL Data Management

**Course Instructor**: Sunil Sahoo

**Lab Session**: 04

**Activity**: This lab focuses on exploring **MongoDB query modifiers** and retrieving data using different operators, projections, sorting, limiting, and skipping. Students will work on two collections—**students** and **employees**—across two levels of practice.

# Learning Objectives (Los)

**LO1:** Understand MongoDB find() syntax and query filters**.**

**LO2:** Apply query modifiers like $gt, $lt, $in, $ne, limit(), and skip().

**LO3:** Perform projection to control output fields.

**LO4:** Combine query filters with sorting and pagination for efficient retrieval.

## **Qualifiers in MongoDB**

Qualifiers in MongoDB are mechanisms that precisely target **specific fields, nested fields, or array elements** within documents during queries, projections, and updates. They enhance query precision, allowing developers to interact with complex, nested data structures without retrieving or modifying unnecessary information.

### **1. Purpose and Power of Qualifiers**

* **Field Targeting:** Qualifiers allow you to pinpoint nested fields using **dot notation** (e.g., "address.city"), enabling operations on deeply embedded data.
* **Array Element Updates:** Using the $ positional operator and other array qualifiers ($[], $[<identifier>]), you can directly update only the matching element(s) in an array.
* **Projection Control:** Qualifiers let you include or exclude specific fields in the query output, reducing payload size and improving efficiency.

### **2. Key Advantages**

* **Precision:** Work with exactly the data you need, no more, no less.
* **Efficiency:** Avoids fetching entire documents when only certain fields are needed, minimizing memory and network usage.
* **Flexibility:** Handles nested objects and complex data models without requiring schema restructuring.
* **Fine-Grained Updates:** Update or read only targeted portions of documents instead of replacing entire documents.

### **3. Common Types of Qualifiers**

**Dot Notation:**  
db.students.find({ "address.city": "Bangalore" }) — targets a nested field.

**Positional Operator $:**  
db.students.updateOne({ "scores.type": "math" }, { $set: { "scores.$.marks": 95 } }) — updates the first matching array element.

**All Array Elements $[]:**  
db.students.updateMany({}, { $set: { "scores.$[].passed": true } }) — updates every element in an array.

**Filtered Array Elements $[<identifier>]:**  
db.students.updateMany({}, { $set: { "scores.$[high].bonus": 5 } }, { arrayFilters: [ { "high.marks": { $gte: 90 } } ] })

### **4. Real-World Applications**

* **User Profile Management:** Update only specific parts of a nested settings object without overwriting the entire profile.
* **E-Commerce:** Adjust the quantity or price of a single product in a customer’s order array without touching other items.
* **IoT Monitoring:** Update sensor readings for only certain sensors in a device’s data array.

### **5. Integration with Business Logic**

Qualifiers can directly support application features that require selective updates or queries—such as adjusting one course grade for a student, marking a single notification as read, or updating an order status without reloading all order details.

**Summary:**  
Qualifiers in MongoDB give developers **surgical control** over their data. By targeting specific fields, nested properties, and array elements, qualifiers make queries and updates faster, lighter, and more precise. This capability is essential for applications dealing with complex or nested JSON structures, ensuring both **performance** and **data integrity**.

**CREATE Operation in MongoDB**

**Step 1:** Database and Collection Creation

* + Use the use command followed by the name of the database you want to switch to (or create).
  + If "**lab4**" doesn't exist, MongoDB won't create it immediately, but it will switch the context to that database in the shell.

**Step 2:** Create a new Collection in the database

* + You can create a collection using the createCollection() method

**db.createCollection("students");**

**db.cretaeCollection(“employees”);**

**Step 3:** Insert Sample Data.

db.students.insertMany([

{ Name: "Ali", age: 21,Department: "Computer Science", Grade: 8.5 },

{ Name: "Bindhu", age: 20,Department: "Mathematics", Grade: 7.0 },

{ Name: "Chandan", age: 21,Department: "Computer Science", Grade: 8.4 },

{ Name: "David", age: 22,Department: "Physics", Grade: 6.8 },

{ Name: "Esha", age: 20,Department: "Mathematics", Grade: 8.0 },

{ Name: "Farooq", age: 23, Department: "Physics", Grade: 7.5 },

{ Name: "Gracy", age: 19,Department: "Computer Science", Grade: 7.9 }

]);

db.employees.insertMany([

{emp\_id: 101, name: "Sameer", department: "IT", salary: 65000, experience: 3},

{emp\_id: 102, name: "Saranash", department: "HR", salary: 55000, experience: 2},

{emp\_id: 103, name: "Stella", department: "Sales", salary: 70000, experience: 4},

{emp\_id: 104, name: "John", department: "IT", salary: 60000, experience: 5},

{emp\_id: 105, name: "Sophia", department: "Marketing", salary: 50000, experience: 1}

])

**Step 4: Level 1 MongoDB Qualifier experiments:**

1. *Display only the 'name' and 'grade' of all students.*

db.students.find({},{name:1,grade:1,\_id:0})

1. *Retrieve all students who are older than 20.*

db.students.find({age: {$gt: 20}})

1. *Get all students who belong to the 'ECE' department.*

db.students.find({department: 'ECE'})

1. *Sort students by 'name' in ascending order.*

db.students.find().sort({name: 1})

1. *Sort students by 'age' in descending order.*

db.students.find().sort({age: -1})

1. *Limit the result to first 3 students.*

db.students.find().limit(3)

1. *Skip the first 2 documents and show the rest.*

db.students.find().skip(2)

1. *Find students with grade 'A' or 'B'.*

db.students.find({grade: {$in: ['A', 'B']}})

1. *Find students whose age is between 18 and 22.*

db.students.find({age: {$gte: 18, $lte: 22}})

1. *Project all students with 'name', excluding '\_id'.*

db.students.find({}, {name: 1, \_id: 0})

**Step 5: Level 2 MongoDB Qualifier experiments:**

1. *Display only the 'name' and 'salary' of all employees.*

db.employees.find({}, {name: 1, salary: 1, \_id: 0})

1. *List employees who have more than 2 years of experience.*

db.employees.find({experience: {$gt: 2}})

1. *Find employees with salary between 50,000 and 70,000.*

db.employees.find({salary: {$gte: 50000, $lte: 70000}})

1. *Find employees not in the 'HR' department.*

db.employees.find({department: {$ne: 'HR'}})

1. *Sort employees by 'salary' in descending order.*

db.employees.find().sort({salary: -1})

1. *Show only top 5 highest-paid employees.*

db.employees.find().sort({salary: -1}).limit(5)

1. *Retrieve employees whose names start with the letter 'S'.*

db.employees.find({name: /^S/})

1. *Project only 'emp\_id' and 'department', excluding '\_id'.*

db.employees.find({}, {emp\_id: 1, department: 1, \_id: 0})

1. *Use $in operator to find employees in 'IT' and 'Sales' departments.*

db.employees.find({department: {$in: ['IT', 'Sales']}})

1. *Skip the top 3 salaries and show the rest sorted by salary*.

db.employees.find().sort({salary: -1}).skip(3)

**Lab challenge 4: University Management System**

**Background**: A university and a corporate firm share a MongoDB database to manage both students academic records and employee payrolls. They require quick searches, projection, and sorting for their respective needs.

**Requirements:**

1. The **university** admin wants a report of all students above a certain age, sorted by name, with only necessary details shown.
2. The **HR team** in the corporate firm wants to quickly retrieve top earners, filter employees by departments, and search names starting with specific letters.
3. The system must support **pagination** for large datasets using .skip() and .limit().
4. Queries must return only relevant fields to optimize network usage.

**Real-Time Scenario Example:**

* The Dean requests: “Give me the names of all ECE students older than 20.”
* The HR Manager asks: “Show me the top 5 salaries, but skip the first three for confidentiality.”
* A payroll analyst wants: “List employees in IT and Sales departments along with their IDs, but hide other sensitive details.”

Happy Learning