**Course Code:** CSE 3513

**Course Name:** NoSQL Data Management

**Course Instructor**: Sunil Sahoo

**Lab Session**: 03

**Activity**: Working with MongoDB Aggregation.

# Learning Objectives (Los)

**LO1:** Create a database and collection for managing students records**. LO2:** Insert and update sample documents.

**LO3:** Explore different MongoDB aggregation commands.

**LO4:** Apply aggregation for real-world use cases.

**Aggregate operations** in MongoDB represent powerful tools for processing, transforming, and analyzing large datasets directly within the database. They go beyond simple CRUD by enabling data computation and reshaping, making them essential for reporting, analytics, and real-time insights.

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

* **Data Transformation**: Aggregation pipelines allow you to reshape documents—combining, filtering, grouping, and projecting fields in customized formats.
* **Complex Analysis**: With aggregation, you can perform advanced calculations (averages, sums, counts, etc.) without exporting data to external tools.
* **Multiple Stages**: Data flows through sequential stages like $match, $group, $project, $sort, enabling a clear and modular approach to analytics.

### 2. **Key Advantages**

* **Efficiency**: Aggregations are executed server-side, minimizing data transfer and improving performance.
* **Flexibility**: Works with varied data formats due to MongoDB’s document-oriented model.
* **Real-Time Analytics**: Supports up-to-the-moment calculations, perfect for dashboards, monitoring systems, and live reports.

### 3. **Core Stages in Aggregation Pipelines**

* **$match**: Filters documents based on specified criteria, reducing dataset size early in the pipeline.
* **$group**: Groups documents by a key and performs accumulations like sum, avg, max, min.
* **$project**: Reshapes the document by including, excluding, or computing fields.
* **$sort**: Orders results for easier interpretation.
* **$count**: Quickly counts documents after filtering.

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

* **Business Reporting**: Calculate sales by region, average customer spend, or high-performing product categories.
* **Monitoring & Alerts**: Track active users, peak loads, or abnormal activities in real time.
* **Data Cleaning**: Extract only required fields and filter irrelevant data for external processing.

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

* Aggregations can directly support application features—like leaderboards, monthly summaries, or trend visualizations—without extra processing layers.

**Summary**:  
Aggregation operations in MongoDB act as a built-in analytics engine, enabling developers to filter, group, transform, and calculate directly on stored data. Their ability to handle complex data transformations in a single query makes them indispensable for applications that require high-performance analytics, dynamic reporting, and real-time insights—without moving data outside the database.

**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 "**lab3**" 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");**

**Step 3:** Insert Sample Data.

db.students.insertMany([

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

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

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

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

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

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

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

]);

**Step 4:** Update Data.

// Update Farooq’s grade

db.students.updateOne(

{ Name: "Farooq" },

{ $set: { Grade: 8.3 } }

);

// Increase all Physics students' grades by 0.5

db.students.updateMany(

{ Department: "Physics" },

{ $inc: { Grade: 0.5 } }

);

**Step 5: Aggregation Commands.**

1. ***Count total students***

db.students.aggregate([{ $count: "totalStudents" }]);

1. ***Average grade of students***

db.students.aggregate([{ $group: { \_id: null, averageGrade: { $avg: "$Grade" } } }]);

1. ***Group by department with total count***

db.students.aggregate([{ $group: { \_id: "$Department", totalStudents: { $sum: 1 } } }]);

1. ***Average grade per department***

db.students.aggregate([{ $group: { \_id: "$Department", avgGrade: { $avg: "$Grade" } } }]);

1. ***Maximum grade***

db.students.aggregate([{ $group: { \_id: null, maxGrade: { $max: "$Grade" } } }]);

1. ***Departments with average grade > 7***

db.students.aggregate([

{ $group: { \_id: "$Department", avgGrade: { $avg: "$Grade" } } },

{ $match: { avgGrade: { $gt: 7 } } }

]);

1. ***Project names and grades only***

db.students.aggregate([{ $project: { \_id: 0, Name: 1, Grade: 1 } }]);

1. ***Sort students by grade (descending)***

db.students.aggregate([{ $sort: { Grade: -1 } }]);

1. ***Total grade by department (descending)***

db.students.aggregate([

{ $group: { \_id: "$Department", totalGrade: { $sum: "$Grade" } } },

{ $sort: { totalGrade: -1 } }

]);

1. ***High achievers (grade > 7.5) grouped by department***

db.students.aggregate([

{ $match: { Grade: { $gt: 7.5 } } },

{ $group: { \_id: "$Department", highAchievers: { $sum: 1 } } }

]);

**Lab challenge 3: Banking Management System**

# Background:

A **Banking Management System** needs to handle:

* Sorting customer details.
* Searching for customers with a specific loan.
* Sorting customers by account balance for credit card eligibility.
* Supporting scalable analytics using aggregation.

### **Requirements**

1. Create a customers collection.
2. Insert at least **10 customer records** with fields:
   * name, accountNumber, balance, loanType, loanAmount.
3. Implement aggregations to:
   * Find customers with a specific loan type.
   * Sort customers by balance in descending order.
   * Count how many customers are eligible for a premium credit card (balance > 50000).
4. Test filtering + sorting combinations for real-time queries.

### **Real-Time Scenario**

In a real banking app:

* Managers can instantly find top balance holders.
* Loan officers can filter customers by loan type.
* The system can generate department/branch-wise reports in real time.

Happy Learning