MapReduce is a programming model used for processing large datasets in parallel across distributed clusters. In MongoDB, it is used to perform operations on data within collections in a way that can efficiently scale.

* **Map**: The map phase processes each document and transforms it into key-value pairs (or intermediate results).
* **Reduce**: The reduce phase processes the output of the map phase, aggregates or combines the results, and returns the final outcome.

#### **Flow of MapReduce**

1. **Map Phase**:
   * MongoDB applies a map function to each document in the collection.
   * The map function emits key-value pairs.
2. **Shuffle Phase**:
   * MongoDB groups the key-value pairs by key and sends them to the reducers.
3. **Reduce Phase**:
   * MongoDB applies a reduce function to each group of key-value pairs.
   * The reduce function aggregates the results (like summing or averaging values).

Q :Write a MapReduce program to compute the average grade of each student from a large dataset containing student IDs, subjects, and grades. Additionally, identify the highest and lowest grades for each subject.  
  
// Create or switch to the "school" database

use school

// Create the "grades" collection with dummy data

db.grades.insertMany([

{ studentId: 1, subject: 'Math', grade: 85 },

{ studentId: 1, subject: 'English', grade: 90 },

{ studentId: 2, subject: 'Math', grade: 78 },

{ studentId: 2, subject: 'English', grade: 92 },

{ studentId: 3, subject: 'Math', grade: 95 },

{ studentId: 3, subject: 'English', grade: 87 },

{ studentId: 3, subject: 'Science', grade: 90 },

{ studentId: 4, subject: 'Math', grade: 70 },

{ studentId: 4, subject: 'Science', grade: 80 },

{ studentId: 4, subject: 'English', grade: 85 }

]);

/INPUT QUERY :   
db.grades.mapReduce(

function() {

emit(this.student\_id, this.grade);

},

function(student\_id, grades) {

var total = Array.sum(grades);

var count = grades.length;

var avg = total / count;

return { average: avg };

},

{

out: "student\_averages"

}

);

// OUTPUT QUERY :  
db.student\_averages.find()  
  
  
db.grades.mapReduce(...): This is the MongoDB **MapReduce** operation, applied to the grades collection.  
This map function processes each document in the grades collection.

**this.student\_id**: For each document, the student\_id is emitted as the key.

**this.grade**: The grade value for that document is emitted as the value.

**emit()**: This function is used to emit key-value pairs. For every document, a key-value pair (student\_id, grade) is emitted.

* The reduce function is executed for each group of student\_id.
* **grades**: This is an array containing all grades associated with the current student\_id (grouped by Map).
* **Array.sum(grades)**: Calculates the total of the grades array.
* **grades.length**: Calculates the count of grades for the student.
* **avg = total / count**: Computes the average grade.
* **return { average: avg }**: Returns an object with the average grade for the student.

INPUT QUERY :

db.grades.mapReduce(

function() {

emit(this.subject, { minGrade: this.grade, maxGrade: this.grade });

},

function(subject, values) {

var result = {

minGrade: values[0].minGrade,

maxGrade: values[0].maxGrade

};

values.forEach(function(value) {

if (value.minGrade < result.minGrade) {

result.minGrade = value.minGrade;

}

if (value.maxGrade > result.maxGrade) {

result.maxGrade = value.maxGrade;

}

});

return result;

},

{

out: "subject\_grades"

}

);

OUTPUT QUERY :  
db.subject\_grades.find()

db.grades.mapReduce(...): This is the MapReduce operation applied to the grades collection.

**Purpose**: This map function processes each document in the grades collection.

**this.subject**: The subject is emitted as the key.

**{ minGrade: this.grade, maxGrade: this.grade }**: The value is an object containing the grade as both minGrade and maxGrade.

**emit()**: This function emits a key-value pair where the key is the subject, and the value is an object with the grade as both the minimum and maximum grade for that subject.

**Purpose**: This reduce function processes each subject group and aggregates the minGrade and maxGrade.

**values**: This is an array of objects with minGrade and maxGrade for each document emitted by the map function.

**Initial values**: result.minGrade is set to the first document's minGrade, and result.maxGrade is set to the first document's maxGrade.

**Loop**: The values.forEach() loop checks every document in the group:

* If a minGrade is lower than the current minGrade, it updates the minGrade.
* If a maxGrade is higher than the current maxGrade, it updates the maxGrade.

**return result**: Returns the final object containing the minimum and maximum grades for the subject.

**Problem Statement -2**

Write a MapReduce program to analyze a large dataset of banking transactions. The program should compute the **total transaction amount per customer**, as well as identify the **highest and lowest transaction amounts for each customer**. This analysis will help the bank understand individual customer spending patterns and detect potential anomalies. Assumne neccesary field in collection.   
  
use banking\_db;

db.transactions.insertMany([

{ customer\_id: 'C1', transaction\_amount: 1500 },

{ customer\_id: 'C1', transaction\_amount: 3000 },

{ customer\_id: 'C1', transaction\_amount: 500 },

{ customer\_id: 'C2', transaction\_amount: 2500 },

{ customer\_id: 'C2', transaction\_amount: 4500 },

{ customer\_id: 'C2', transaction\_amount: 1000 },

{ customer\_id: 'C3', transaction\_amount: 2000 },

{ customer\_id: 'C3', transaction\_amount: 750 },

{ customer\_id: 'C3', transaction\_amount: 1200 },

{ customer\_id: 'C4', transaction\_amount: 800 },

{ customer\_id: 'C4', transaction\_amount: 3500 }

]);

db.transactions.mapReduce(

function() {

emit(this.customer\_id, { totalAmount: this.transaction\_amount, minAmount: this.transaction\_amount, maxAmount: this.transaction\_amount });

},

function(customer\_id, values) {

var result = {

totalAmount: 0,

minAmount: values[0].minAmount,

maxAmount: values[0].maxAmount

};

values.forEach(function(value) {

result.totalAmount += value.totalAmount;

if (value.minAmount < result.minAmount) {

result.minAmount = value.minAmount;

}

if (value.maxAmount > result.maxAmount) {

result.maxAmount = value.maxAmount;

}

});

return result;

},

{

out: "customer\_transaction\_summary"

});

db.customer\_transaction\_summary.find().sort({ \_id: 1 });

**Problem Statement -3**

Write a MapReduce program to analyze a large dataset of e-commerce sales. The program should calculate the total number of purchases per customer and determine the maximum and minimum purchase values made by each customer. This will help the e-commerce platform gain insights into customer purchase behavior and identify potential loyal customers or high spenders.

use ecommerce\_db;

db.sales.insertMany([

{ customer\_id: 'C1', purchase\_value: 100 },

{ customer\_id: 'C1', purchase\_value: 250 },

{ customer\_id: 'C1', purchase\_value: 150 },

{ customer\_id: 'C2', purchase\_value: 300 },

{ customer\_id: 'C2', purchase\_value: 200 },

{ customer\_id: 'C2', purchase\_value: 400 },

{ customer\_id: 'C3', purchase\_value: 450 },

{ customer\_id: 'C3', purchase\_value: 120 },

{ customer\_id: 'C3', purchase\_value: 600 },

{ customer\_id: 'C4', purchase\_value: 800 },

{ customer\_id: 'C4', purchase\_value: 500 },

]);

db.sales.mapReduce(

function() {

emit(this.customer\_id, { count: 1, minPurchase: this.purchase\_value, maxPurchase: this.purchase\_value });

},

function(customer\_id, values) {

var result = {

count: 0,

minPurchase: values[0].minPurchase,

maxPurchase: values[0].maxPurchase

};

values.forEach(function(value) {

result.count += value.count; // Count total purchases

if (value.minPurchase < result.minPurchase) {

result.minPurchase = value.minPurchase;

}

if (value.maxPurchase > result.maxPurchase) {

result.maxPurchase = value.maxPurchase;

}

});

return result;

},

{

out: "customer\_purchase\_summary"

});

db.customer\_purchase\_summary.find().sort({ \_id: 1 });