****

FACULTY OF ENGINEERING AND TECHNOLOGY

BACHELOR OF TECHNOLOGY

**BIG DATA ANALYSIS**

**(203105348)**

7th SEMESTER

7A13

**NAME**: **VISHAL SURESH UMAVANE**

**ROLL NO.: 32**

**ENROLLMENT NUMBER**: **210306105192**

**CERTIFICATE**

*This is to certify that**Mr.* ***Vishal Suresh Umavane*** *with Enrolment no* ***210306105192*** *has successfully completed his laboratory experiments in the Big data analytics (203105348) from the department of* ***B.Tech CSE*** *during the academic year* ***2024-25***

**

Date of Submission:.........................

Staff In charge:.............................

Head Of Department:...........................................

# INDEX

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Sr. No | Experiment Title | Page No | | Date of Start | Date of Completion | Sign | Remarks |
| From | To |
| 1. | To understand the overall programming architecture using Map Reduce API. |  |  |  |  |  |  |
| 2. | Write a program of Word Count in Map Reduce over HDFS. |  |  |  |  |  |  |
| 3. | Basic CRUD operations in MongoDB. |  |  |  |  |  |  |
| 4. | Store the basic information about students such as roll no, name, date of birth, and address of student using various collection types such as List, Set and Map. |  |  |  |  |  |  |
| 5. | Basic commands available for the Hadoop Distributed File System. |  |  |  |  |  |  |
| 6. | Basic commands available for HIVE Query Language. |  |  |  |  |  |  |
| 7. | Basic commands of HBASE Shell. |  |  |  |  |  |  |
| 8. | Creating the HDFS tables and loading them in Hive and learn joining of tables in Hive. |  |  |  |  |  |  |

**PRACTICAL 1**

**Aim:** To Understand the overall programing architecture using Map Reduce API.

**Solution:**

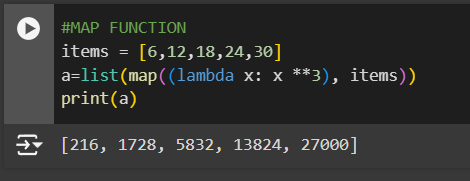
* **MapReduce** and **HDFS** are the two major components of Hadoop which makes it so powerful and efficient to use.
* Reduce is a programming model used for efficient processing in parallel over large data-sets in a distributed manner.
* The data is first split and then combined to produce the final result.
* The libraries for MapReduce is written in so many programming languages with various different-different optimizations.
* The purpose of MapReduce in Hadoop is to Map each of the jobs and then it will reduce it to equivalent tasks for providing ess overhead over the cluster network and to reduce the processing power.
* MapReduce task is mainly divided into two phases Map Phase and Reduce Phase.
* map ( ), filter(), and reduce() in Python.
* These functions are most commonly used with Lambda function.

### Map() Function:

A map function executes certain instructions or functionality provided to it on every item of an iterable. "The iterable could be a list, tuple, set, etc.

Syntax:

map(function, iterable)



The Map() function passes each element in the list to a lambda function and return the mapped object.

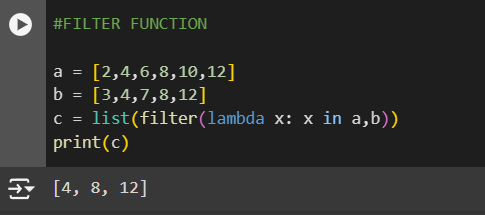
## Filter() Function:

A Filter function in Python tests a specific user defined condition for a function and returns an iterable for the elements and values that satisfy the condition or, in other words, return True.

Syntax:

Filter(function, iterable)

Example:



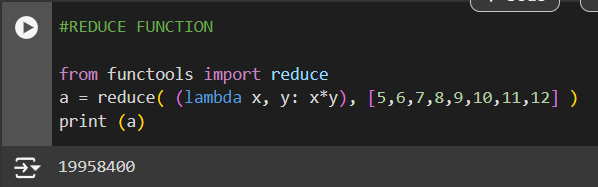
## Reduce() Function:

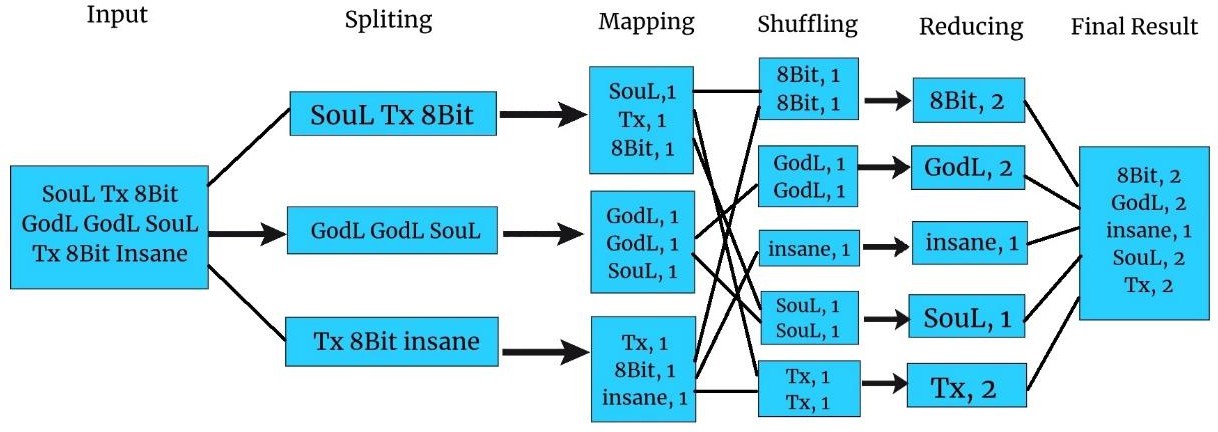
Reduce functions apply a function to every item of an iterable and gives back a single value as a resultant.

We have to import to reduce function from Functools module using the statement. Syntax:

Reduce(function, iterable)

Example:





**PRACTICAL 2**

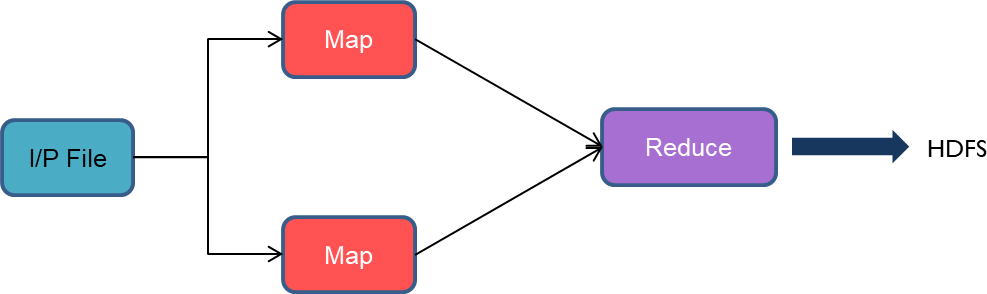
**Aim:** Write a program of Word Count in Map Reduce over HDFS.

### Description:

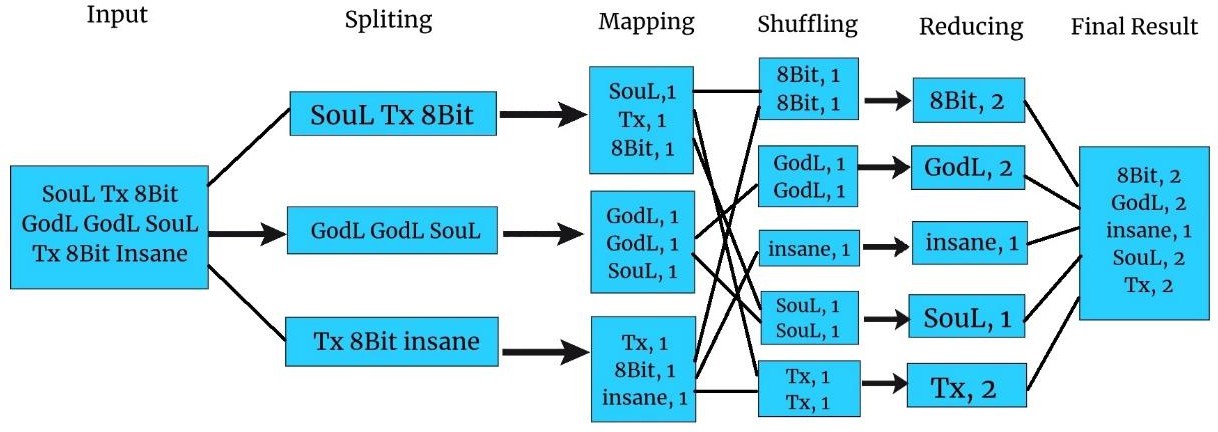
MapReduce is a framework for processing large datasets using a large number of computers (nodes), collectively referred to as a cluster. Processing can occur on data stored in a file system (HDFS).A method for distributing computation across multiple nodes.Each node processes the data that is stored at that node.

Consists of two main phases Mapper Phase

Reduce phase



Input data set is split into independent blocks – processed in parallel. Each input split is converted in Key Value pairs. Mapper logic processes each key value pair and produces and intermediate key value pairs based on the implementation logic. Resultant key value pairs can be of different type from that of input key value pairs. The output of Mapper is passed to the reducer. Output of Mapper function is the input for Reducer. Reducer sorts the intermediate key value pairs. Applies reducer logic upon the key value pairs and produces the output in desired format.Output is stored in HDFS



## Code:

import urllib.request import random

from operator import itemgetter

current\_word = {} current\_count = 0

story = ['http://sixty](http://sixty-north.com/c/t.txt%27)-[north.com/c/t.txt'](http://sixty-north.com/c/t.txt%27) request = urllib.request.Request(story)

def map\_function(line): words = line.split()

return [(word.lower(), 1) for word in words]

def reduce\_function(mapped\_values): word\_counts = {}

for word, count in mapped\_values: if word in word\_counts:

word\_counts[word] += count else:

word\_counts[word] = count return word\_counts

def map\_reduce():

global current\_word, current\_count

with urllib.request.urlopen(request) as response: text = response.read().decode('utf-8')

mapped = []

for line in text.split('\n'): mapped.extend(map\_function(line))

random.shuffle(mapped)

reduced = reduce\_function(mapped)

sorted\_counts = sorted(reduced.items(), key=itemgetter(1), reverse=True) if sorted\_counts:

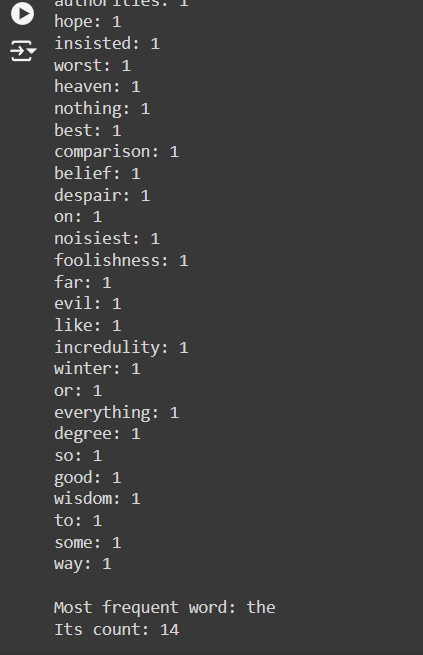
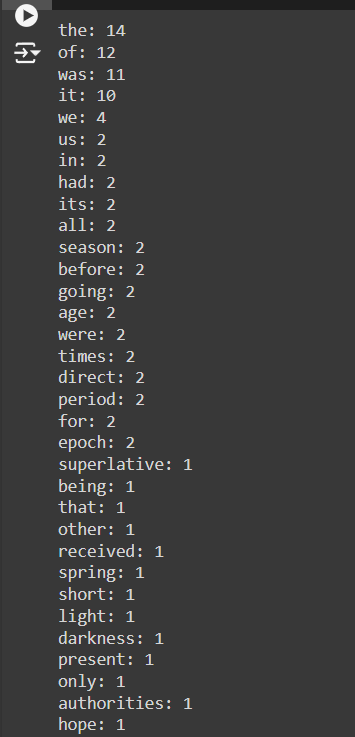
current\_word, current\_count = sorted\_counts[0] return sorted\_counts

result = map\_reduce()

for word, count in result: print(f"{word}: {count}")

print(f"\nMost frequent word: {current\_word}") print(f"Its count: {current\_count}")

**Output:**



# PRACTICAL 3

**Aim:** To study Basic CRUD operations in MongoDB.

### Description: CRUD operation does:

* **Create**
  + Adds new records or data to the database.
  + Example: Registering a new user account.
  + SQL Command: INSERT INTO table\_name (columns) VALUES (values);
  + HTTP Method: POST
* **Read**
  + Retrieves existing records or data from the database.
  + Example: Fetching user details or listing all users.
  + SQL Command: SELECT columns FROM table\_name WHERE condition;
  + HTTP Method: GET
* **Update**
  + Modifies existing records or data in the database.
  + Example: Updating user profile information.
  + SQL Command: UPDATE table\_name SET column1 = value1, column2 = value2 WHERE condition;
  + HTTP Method: PUT or PATCH
* **Delete**
  + Removes records or data from the database.
  + Example: Deleting a user account.
  + SQL Command: DELETE FROM table\_name WHERE condition;
  + HTTP Method: DELETE

**Program:** db.createCollection("employees"); db.employees.insertMany([

]);

{empId: 1, name: 'Clark', dept: 'Sales' },

{empId: 2, name: 'Dave', dept: 'Accounting' },

{empId: 3, name: 'Ava', dept: 'Sales' }

db.employees.find({dept: 'Sales'});

db.employees.insert({empId: 4, name: 'Raja', dept: 'marketing' }) db.employees.update({name:'Raja'},{$set:{name:'Alise'}}); db.employees.find({dept: 'marketing'}); db.employees.remove({dept:'Accounting'});

db.createCollection("orders") db.orders.insertMany( [

{ id: 0, name: "Pepperoni", size: "small", price: 19, quantity: 10, date: ISODate( "2021-03-13T08:14:30Z" ) },

{ id: 1, name: "Pepperoni", size: "medium", price: 20, quantity: 20, date : ISODate( "2021-03-13T09:13:24Z" ) },

{ id: 2, name: "Pepperoni", size: "large", price: 21,

quantity: 30, date : ISODate( "2021-03-17T09:22:12Z" ) },

{ id: 3, name: "Cheese", size: "small", price: 12,

quantity: 15, date : ISODate( "2021-03-13T11:21:39.736Z" ) },

{ id: 4, name: "Cheese", size: "medium", price: 13,

quantity:50, date : ISODate( "2022-01-12T21:23:13.331Z" ) },

{ id: 5, name: "Cheese", size: "large", price: 14,

quantity: 10, date : ISODate( "2022-01-12T05:08:13Z" ) },

{ id: 6, name: "Vegan", size: "small", price: 17,

quantity: 10, date : ISODate( "2021-01-13T05:08:13Z" ) },

{ id: 7, name: "Vegan", size: "medium", price: 18,

quantity: 10, date : ISODate( "2021-01-13T05:10:13Z" ) }

] )

db.orders.find({size: "medium"});

db.orders.insert({id: 9, name: "Vegan", size: "medium", price: 8, quantity: 5, date : ISODate( "2021-01-22T05:10:13Z" )})

db.orders.updateMany({name:'Vegan'},{$set:{name:'Veg'}}) db.orders.find({name: 'Veg'}); db.orders.remove({name:'Pepperoni'})

db.orders.find({ $and: [ {name: 'Veg'}, { size: "small"} ] }) db.orders.find()

**Output:**



**Practical 4**

**Aim:** Store the basic information about students such as roll no, name, date of birth and address of student using various collection types such as List, Set and Map.

**Description :**

**Using List**

* **Definition:** An ordered collection that allows duplicates and maintains the order of insertion.
* **Use Case:** When the order of students matters or you need to access them by their position.
* **Example:** Storing students in the order they enrolled or taking attendance.

**Using Set**

* **Definition:** An unordered collection that does not allow duplicates.
* **Use Case:** Ensuring each student is unique and preventing duplicate entries.
* **Example:** Storing unique student records to avoid duplicate roll numbers.

**Using Map**

* **Definition:** A collection of key-value pairs where each key is unique.
* **Use Case:** Associating student roll numbers (keys) with their details (values) for quick look-up.
* **Example:** Using roll numbers as keys to retrieve student information efficiently.

**Program 1 :**

use studentdb;

db.createCollection("student");

// Insert multiple documents using insertMany with an array

db.student.insertMany([

{

no: 1,

name: "Vishal",

dob: "19-11-2002",

e\_mail: "vumavane@gmail.com",

phone: "7039144205",

address: {

building: "1234",

street: "main road",

zipcode: "421306"

},

Branch: "CSE",

marks: [50, 70, 60]

},

{

no: 2,

name: "Harsh",

dob: "17-07-2003",

e\_mail: "harshsingh@gmail.com",

phone: "8208629356",

address: {

building: "5678",

street: "main road",

zipcode: "431511"

},

Branch: "CSE",

marks: [45, 20, 80]

},

{

no: 3,

name: "Akash",

dob: "30-09-2003",

e\_mail: "akashpanchal@gmail.com",

phone: "9890898563",

address: {

building: "1256",

street: "main road",

zipcode: "431511"

},

Branch: "CSE",

marks: [59, 89, 68]

},

{

no: 4,

name: "Vidya",

dob: "15-04-2003",

e\_mail: "vidya85@gmail.com",

phone: "9698741235",

address: {

building: "3478",

street: "main road",

zipcode: "431511"

},

Branch: "CSE",

marks: [53, 29, 33]

},

{

no: 5,

name: "Sahil",

dob: "13-05-2003",

e\_mail: "sahil168@gmail.com",

phone: "9642317893",

address: {

building: "2525",

street: "ain road",

zipcode: "431511"

},

Branch: "CSE",

marks: [61, 20, 66]

}

]);

db.student.find()

db.student.update({no:1},{$set:{name:"Rishikesh"}})

db.student.find({name:/^R/})

db.student.find({name:/l$/})

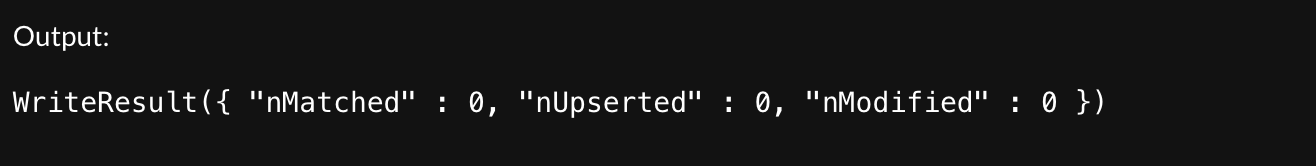
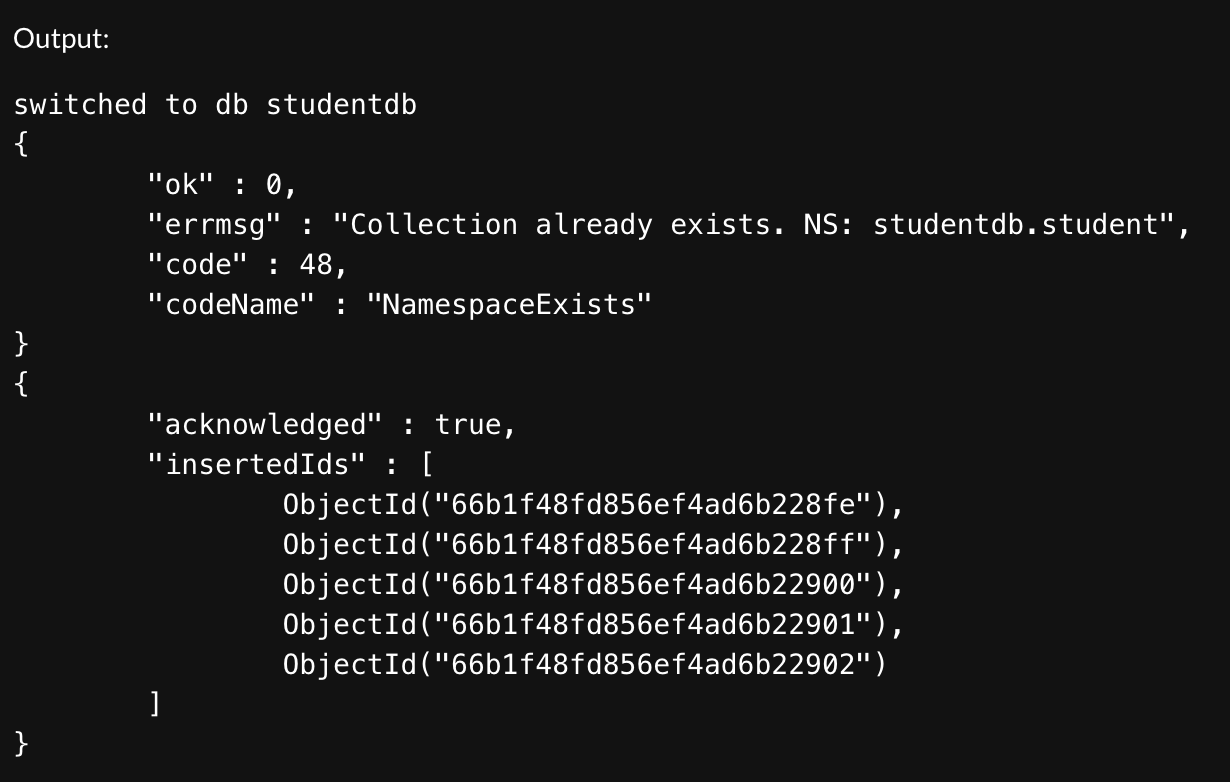
db.student.find({name:/S/})

db.student.count()

db.studnet.find().forEach(function(myDoc) {print("name: Vidya "+myDoc.name);})

db.student.find().limit(2);

db.student.find().sort({"name": -1})

Output :

**Program-2**

db.grades.insertMany([

{\_id: 1, quizzes: [5,6,7]},

{\_id: 2, quizzes: []},

{\_id: 3, quizzes: [3,8,9]}

])

db.grades.aggregate(

[

{

$project:

{

adjustedGrades:

{

$map:

{

input: "$quizzes",

as: "grade",

in: {$add: ["$$grade",2]}

}

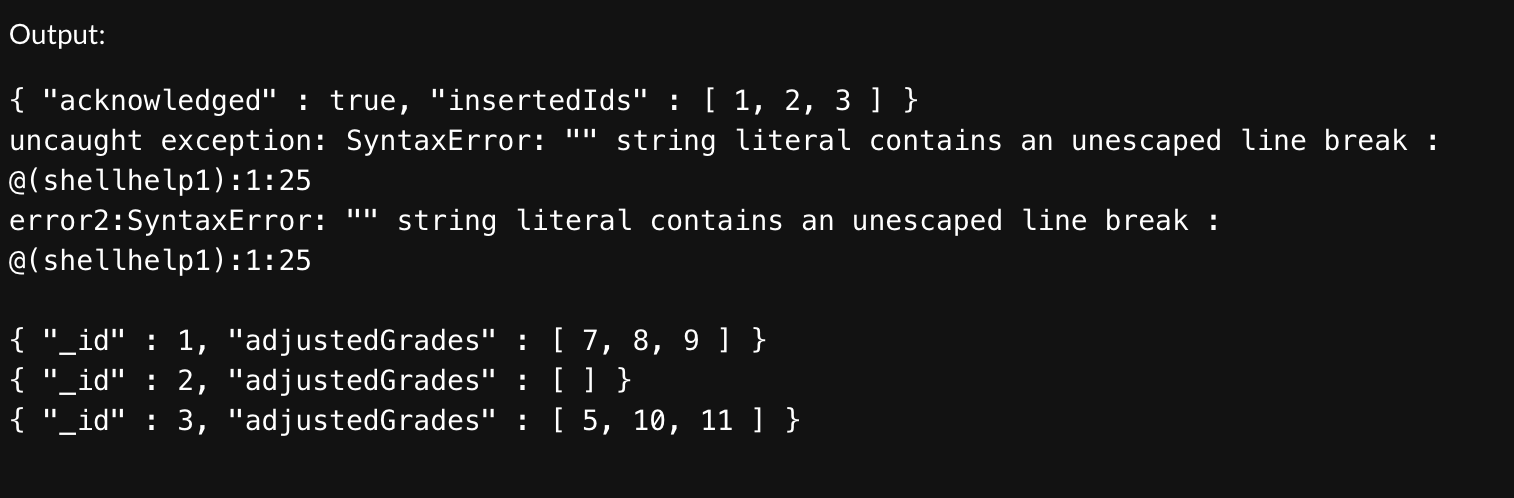
}

}

}

]

)



**Program-3**

db.scores.insertMany([

{\_id: 1, student: "Maya", homework: [10,5,10], quiz: [10,8],extraCredit: 0},

{\_id: 2, student: "Ryan", homework: [5,6,5], quiz: [8,8],extraCredit: 8}

])

db.scores.aggregate([

{

$set:{

totalHomework:{$sum: "$homework"},

totalQuiz:{$sum:"$quiz"}

}

},

{

$set:{

totalScore:{$add:["$totalHomework","$totalQuiz","$extraCredit"]}

}

}

])

