**Assignment No:8**

**Title:** Implement Map reduces operation with suitable example using MongoDB.

**Problem Statement:** Implement the MapReduce operation in MongoDB by defining map and reduce functions, executing them on a dataset, and demonstrating the process with a suitable example.

**Objective:** To gain knowledge of NoSQL databases for processing unstructured data.

**Outcome:** Use NoSQL databases for processing unstructured data.

**Tools Required:** Ubuntu OS, MongoDB.

**Theory:**

**1. MongoDB-MapReduce**

MapReduce is a data processing model used in MongoDB for aggregating large datasets efficiently. It consists of two primary functions:

* **Map Function:** Processes input documents and emits key-value pairs.
* **Reduce Function:** Takes the output from the Map function, groups the values by key, and processes them to generate aggregated results.

**Map-reduce** is a data processing paradigm for condensing large volumes of data into useful aggregated results. MongoDB uses **mapReduce** command for map-reduce operations. MapReduce is generally used for processing large data sets.

* 1. **Syntax**

|  |
| --- |
| >db.collection.mapReduce(  function() {emit(key,value);}, //map function  function(key,values) {return reduceFunction}, { //reduce function  out: collection,  query: document,  sort: document,  limit: number  }  ) |

The map-reduce function first queries the collection, then maps the result documents to emit key-value pairs, which is then reduced based on the keys that have multiple values.

In the above syntax −

* **map** is a javascript function that maps a value with a key and emits a key-value pair
* **reduce** is a javascript function that reduces or groups all the documents having the same key
* **out** specifies the location of the map-reduce query result
* **query** specifies the optional selection criteria for selecting documents
* **sort** specifies the optional sort criteria
* **limit** specifies the optional maximum number of documents to be returned

**Step 1: Create a Sample Collection:** Create a sales collection with product sales data:

|  |
| --- |
| db.sales.insertMany([  { "\_id": 1, "product": "Laptop", "quantity": 2, "price": 50000 },  { "\_id": 2, "product": "Mobile", "quantity": 5, "price": 20000 },  { "\_id": 3, "product": "Laptop", "quantity": 1, "price": 50000 },  { "\_id": 4, "product": "Tablet", "quantity": 3, "price": 30000 }  ]); |

**Step 2: Define the Map Function:** The map function emits key-value pairs, where the key is the product name and the value is the total revenue (quantity × price).

|  |
| --- |
| var mapFunction = function() { emit(this.product, this.quantity \* this.price); }; |

**Step 3: Define the Reduce Function:** The reduce function sums up all revenue values for each product.

|  |
| --- |
| var reduceFunction = function(key, values) { return Array.sum(values); }; |

**Step 4: Execute the MapReduce Operation:** Run the MapReduce function on the sales collection and store the results in a new collection named total\_sales\_by\_product.

|  |
| --- |
| db.sales.mapReduce( mapFunction, reduceFunction, { out: "total\_sales\_by\_product" } ); |

**Step 5: View the Results:** Retrieve the computed total sales per product:

|  |
| --- |
| db.total\_sales\_by\_product.find().pretty(); |

**Output:**

|  |
| --- |
| { "\_id": "Laptop", "value": 150000 }  { "\_id": "Mobile", "value": 100000 }  { "\_id": "Tablet", "value": 90000 } |

**Conclusion:**

We have successfully implemented MongoDB aggregation and indexing operations.