Algorithm Design Using MapReduce

1. Write a map only algorithm which will read the original dataset as input and filter out all the records which have event_epoch_time, user_id, device_id, user_agent as NULL.

MAP_ELIMINATE_RECORD_WITH_NULL (Key, Value)

```
START
   LS = Split (Value, "\t")
   EVENT_EPOCH_TIME = LS[1]
   USER_ID = LS[2]
   DEVICE_ID = LS[3]
   USER_AGENT = LS[4]
   IF ( EVENT_EPOCH_TIME != NULL  AND  USER_ID != NULL  AND
   DEVICE_ID != NULL AND USER_AGENT != NULL )
      Write (Key, Value)
END
```

MAP DOCUMENTATION FOR QUESTION 1

➔ A transaction record is stored in **Value.** The **Split** function extracts **12** values out of the record using a **tab** delimiter and stores these values in a **List LS.**

➔ The record is filtered out i.e. the record is not written as output if any of the following values is **NULL: EVENT_EPOCH_TIME, USER_ID, DEVICE_ID, USER_AGENT.**

2. An algorithm to read the user agent and extract OS Version and Platform from it.

MAP_EXTRACT_OS_VERSION_AND_PLATFORM (Key, Value)

```
START
   LS = Split (Value, "\t")
   USER_AGENT = LS[4]
   NEW_LS = Split (USER_AGENT,":")
   PLATFORM = NEW_LS[1]
   OS_VERSION = NEW_LS[2]
   Write (USER_AGENT, {PLATFORM, OS_VERSION})
END
```

MAP DOCUMENTATION FOR QUESTION 2

➔ A transaction record is stored in **Value.** The **Split** function extracts **12** values out of the record using a **tab** delimiter and stores these values in a **List LS.**

➔ The value for **USER_AGENT** is available at **LS[4].** The Split function is used with "**:**" as delimiter to extract **Platform** and **OS Version** information.

➔ The record is written using a **Write** function with **USER_AGENT** as **Key** and {**PLATFORM**, **OS_VERSION**} as **ValueList.**

3. Assume there is a predefined method named getCounter(String name) which takes a name as the parameter and creates a global counter variable of the same name if already not created. This global counter variable is accessible to all the map tasks. To increment the value of a counter the method to be used is incrementBy(integer num). Here “num” is the number by which we want to increment the global variable. So the syntax to increment the value of a counter is:
getCounter(“Orders”).incrementBy(1)
Using the above info write algorithms to perform below-mentioned tasks:

I. Find out the number of veg and non-veg pizzas sold

```
MAP_TOTAL_VEG_AND_NON_VEG_PIZZA_SOLD (Key, Value)
START
   LS = Split (Value, “\t”)
   IS_VEG = LS[12]
   ORDER_EVENT = LS[11]
   IF ( IS_VEG == “Y”  AND ORDER_EVENT == “Delivered” )
      getCounter(“VEG_PIZZA_SOLD”).incrementBy(1)
   ELSE IF (IS_VEG == “N” AND ORDER_EVENT == “Delivered”)
      getCounter(“NON-VEG_PIZZA_SOLD”).incrementBy(1)
END
```

MAP DOCUMENTATION FOR QUESTION 3.I

➔ A transaction record is stored in **Value.** The **Split** function extracts **12** values out of the record using a **tab** delimiter and stores these values in a **List LS.**

➔ The information whether a Pizza is Veg or Non-Veg is available at **LS[12].** The information on **ORDER_EVENT** is available at **LS[11].**

➔ **02** global counter variables **VEG_PIZZA_SOLD** and **NON-VEG_PIZZA_SOLD** are created.

➔ The **VEG_PIZZA_SOLD** global counter variable is inremented if **IS_VEG** is **Y** and **ORDER_EVENT** is **Delivered.**

➔ The **NON-VEG_PIZZA_SOLD** global counter variable is inremented if **IS_VEG** is **N** and **ORDER_EVENT** is **Delivered.**

➔ It is presumed that the **ORDER_EVENT** could have multiple values like **Placed, Initiated, On The Way, Cancelled** And **Delivered.** The **Delivered** status will ensure that the Pizza is sold as a Delivered status cannot be changed to **Cancelled.** At all other stages, the **ORDER_EVENT** could be changed to **Cancelled.**

II. Find out the size wise distribution of pizzas sold

```
MAP_TOTAL_PIZZA_SOLD_CATEGORIZED_BY_SIZE (Key, Value)
START
   LS = Split (Value, “\t”)
   SIZE = LS[7]
   ORDER_EVENT = LS[11]
   IF ( SIZE == “S” AND ORDER_EVENT == “Delivered” )
      getCounter(“SMALL_PIZZA_SOLD”).incrementBy(1)
   ELSE IF ( SIZE == “R” AND ORDER_EVENT == “Delivered” )
      getCounter(“REGULAR_PIZZA_SOLD”).incrementBy(1)
   ELSE IF ( SIZE == “M” AND ORDER_EVENT == “Delivered” )
      getCounter(“MEDIUM_PIZZA_SOLD”).incrementBy(1)
   ELSE IF ( SIZE == “L” AND ORDER_EVENT == “Delivered” )
      getCounter(“LARGE_PIZZA_SOLD”).incrementBy(1)
END
```

MAP DOCUMENTATION FOR QUESTION 3.II

➔ A transaction record is stored in **Value.** The **Split** function extracts **12** values out of the record using a **tab** delimiter and stores these values in a **List LS.**

➔ The information regarding the Size of the Pizza is available at **LS[7].** It is presumed that there are **04** Sizes of Pizza. **S** is **Small, R** is **Regular, M** is **Medium** and **L** is **Large.**

➔ The information on **ORDER_EVENT** is available at **LS[11].**

➔ **04** global counter variables **SMALL_PIZZA_SOLD, REGULAR_PIZZA_SOLD, MEDIUM_PIZZA_SOLD** and **LARGE_PIZZA_SOLD** are created.

➔ The **SMALL_PIZZA_SOLD** global counter variable is inremented if the **SIZE** is **S** and **ORDER_EVENT** is **Delivered.**

➔ The **REGULAR_PIZZA_SOLD** global counter variable is inremented if the **SIZE** is **R** and **ORDER_EVENT** is **Delivered.**

➔ The **MEDIUM_PIZZA_SOLD** global counter variable is inremented if the **SIZE** is **M** and **ORDER_EVENT** is **Delivered.**

➔ The **LARGE_PIZZA_SOLD** global counter variable is inremented if the **SIZE** is **L** and **ORDER_EVENT** is **Delivered.**

➔ It is presumed that the **ORDER_EVENT** could have multiple values like **Placed, Initiated, On The Way, Cancelled** And **Delivered.** The **Delivered** status will ensure that the Pizza is sold as a Delivered status cannot be changed to **Cancelled.** At all other stages, the **ORDER_EVENT** could be changed to **Cancelled.**

III. Find out how many cheese burst pizzas were sold

```
MAP_TOTAL_CHEESE_BURST_PIZZA_SOLD (Key, Value)
START
   LS = Split (Value, "\t")
   IS_CHEESE_BURST = LS[6]
   ORDER_EVENT = LS[11]
   IF ( IS_CHEESE_BURST == "Y" AND ORDER_EVENT ==
   "Delivered" )
      getCounter("CHEESE_BURST_PIZZA_SOLD").incrementBy(1)
END
```

MAP DOCUMENTATION FOR QUESTION 3.III

➔ A transaction record is stored in **Value.** The **Split** function extracts **12** values out of the record using a **tab** delimiter and stores these values in a **List LS.**

➔ The information whether a Pizza is **Cheese Burst or Not** is available at **LS[6].**

➔ The information on **ORDER_EVENT** is available at **LS[11].**

➔ A global counter variable **CHEESE_BURST_PIZZA_SOLD** is created.

➔ The **CHEESE_BURST_PIZZA_SOLD** global counter variable is inremented if **IS_CHEESE_BURST** is **Y** and **ORDER_EVENT** is **Delivered.**

➔ It is presumed that the **ORDER_EVENT** could have multiple values like **Placed, Initiated, On The Way, Cancelled** And **Delivered.** The **Delivered** status will ensure that the Pizza is sold as a Delivered status cannot be changed to **Cancelled.** At all other stages, the **ORDER_EVENT** could be changed to **Cancelled.**

IV. Find out how many small cheese burst pizzas were sold. Ideally, the count should be 0 because cheese burst is available for medium and large

```
MAP_TOTAL_SMALL_CHEESE_BURST_PIZZA_SOLD (Key, Value)
START
  LS = Split (Value, “\t”)
  IS_CHEESE_BURST = LS[6]
  SIZE = LS[7]
  ORDER_EVENT = LS[11]
  IF ( IS_CHEESE_BURST == “Y”  AND  SIZE == “S” AND
  ORDER_EVENT == ”Delivered” )
    getCounter(“SMALL_CHEESE_BURST_SOLD”).incrementBy(1)
END
```

MAP DOCUMENTATION FOR QUESTION 3.IV

➔ A transaction record is stored in **Value.** The **Split** function extracts **12** values out of the record using a **tab** delimiter and stores these values in a **List LS.**

➔ The information whether a Pizza is **Cheese Burst or Not** is available at **LS[6].**

➔ The information regarding the Size of the Pizza is available at **LS[7].** It is presumed that there are **04** Sizes of Pizza. **S** is **Small, R** is **Regular, M** is **Medium** and **L** is **Large.**

➔ The information on **ORDER_EVENT** is available at **LS[11].**

➔ A global counter variable **SMALL_CHEESE_BURST_SOLD** is created.

➔ The **SMALL_CHEESE_BURST_SOLD** global counter variable is inremented if **IS_CHEESE_BURST** is **Y, SIZE** is **S** and **ORDER_EVENT** is **Delivered.**

➔ It is presumed that the **ORDER_EVENT** could have multiple values like **Placed, Initiated, On The Way, Cancelled** And **Delivered.** The **Delivered** status will ensure that the Pizza is sold as a Delivered status cannot be changed to **Cancelled.** At all other stages, the **ORDER_EVENT** could be changed to **Cancelled.**

V. Find out the number of cheese burst pizzas whose cost is below Rs 500

```
MAP_TOTAL_CHEESE_BURST_PIZZA_LESS_THAN_RS_500 (Key, Value)
START
   LS = Split (Value, "\t")
   IS_CHEESE_BURST = LS[6]
   PRICE = LS[9]
   IF ( IS_CHEESE_BURST == "Y"  AND  PRICE  < 500)
      getCounter("CHEESE_BURST_LESS_THAN_500").incrementBy(1)
END
```

MAP DOCUMENTATION FOR QUESTION 3.V

➔ A transaction record is stored in **Value.** The **Split** function extracts **12** values out of the record using a **tab** delimiter and stores these values in a **List LS.**

➔ The information whether a Pizza is **Cheese Burst or Not** is available at **LS[6].**

➔ The information regarding the **Price** of a Pizza is available at **LS[9].**

➔ A global counter variable **CHEESE_BURST_LESS_THAN_500** is created.

➔ The **CHEESE_BURST_LESS_THAN_500** global counter variable is inremented if **IS_CHEESE_BURST** is **Y** and **PRICE** is less than **500.**

➔ Unlike other algorithms, we have not considered the **ORDER_EVENT** as the question does not specifically talk about **Sold Pizza.**

4. Assume that the predefined method getCounter does not exist. Write the updated algorithms for the tasks in point-3.

I. Find out the number of veg and non-veg pizzas sold

```
MAP_TOTAL_VEG_AND_NON_VEG_PIZZA_SOLD (Key, Value)
START
   LS = Split (Value, “\t”)
   IS_VEG = LS[12]
   ORDER_EVENT = LS[11]
   IF ( IS_VEG == “Y”  AND ORDER_EVENT == “Delivered” )
      Write (Veg_Pizza_Sold, 1)
   ELSE IF (IS_VEG == “N” AND ORDER_EVENT == “Delivered” )
      Write (Non-Veg_Pizza_Sold, 1)
END
```

MAP DOCUMENTATION FOR QUESTION 4.I

➔ A transaction record is stored in **Value.** The **Split** function extracts **12** values out of the record using a **tab** delimiter and stores these values in a **List LS.**

➔ The information whether a Pizza is Veg or Non-Veg is available at **LS[12].** The information on **ORDER_EVENT** is available at **LS[11].**

➔ If **IS_VEG** is **Y** and **ORDER_EVENT** is **Delivered** then the output is written using a **Write** function. **Veg_Pizza_Sold** is the **Key** and **1** is the **Value.**

➔ If **IS_VEG** is **N** and **ORDER_EVENT** is **Delivered** then the output is written using a **Write** function. **Non-Veg_Pizza_Sold** is the **Key** and **1** is the **Value.**

➔ It is presumed that the **ORDER_EVENT** could have multiple values like **Placed, Initiated, On The Way, Cancelled** And **Delivered.** The **Delivered** status will ensure that the Pizza is sold as a Delivered status cannot be changed to **Cancelled.** At all other stages, the **ORDER_EVENT** could be changed to **Cancelled.**

➔ The output of this **MAP** will be aggregated and provided as input to **REDUCE.**

- For example, the output of the MAP would be (Veg_Pizza_Sold, 1), (Veg_Pizza_Sold, 1),(Veg_Pizza_Sold, 1), (Non-Veg_Pizza_Sold, 1).
- The aggregated result will be (Veg_Pizza_Sold, {1,1,1}) and (Non-Veg_Pizza_Sold, {1})

REDUCE_TOTAL_VEG_AND_NON_VEG_PIZZA_SOLD (Key, ValueList)

```
START
   Count = 0
   FOR i = 1 To ValueList.length
      Count = Count + 1
   Write (Key, Count)
END
```

REDUCE DOCUMENTATION FOR QUESTION 4.I

➔ The input to the **REDUCE** would be the aggregated output of the **MAP.** For example, (Veg_Pizza_Sold, {1,1,1}) and (Non-Veg_Pizza_Sold, {1})

➔ The **REDUCE** will loop through the ValueList and will count the total items in the list.

➔ The output of the **REDUCE** will be Key and Count. For example, (Veg_Pizza_Sold, 3) and (Non-Veg_Pizza_Sold, 1)

II. Find out the size wise distribution of pizzas sold

```
MAP_TOTAL_PIZZA_SOLD_CATEGORIZED_BY_SIZE (Key, Value)
START
  LS = Split (Value, “\t”)
  SIZE = LS[7]
  ORDER_EVENT = LS[11]
  IF ( SIZE == “S” AND ORDER_EVENT == “Delivered” )
    Write (Small_Pizza_Sold, 1)
  ELSE IF ( SIZE == “R” AND ORDER_EVENT == “Delivered” )
    Write (Regular_Pizza_Sold, 1)
  ELSE IF ( SIZE == “M” AND ORDER_EVENT == “Delivered” )
    Write (Medium_Pizza_Sold, 1)
  ELSE IF ( SIZE == “L” AND ORDER_EVENT == “Delivered” )
    Write (Large_Pizza_Sold, 1)
END
```

MAP DOCUMENTATION FOR QUESTION 4.II

➔ A transaction record is stored in **Value.** The **Split** function extracts **12** values out of the record using a **tab** delimiter and stores these values in a **List LS.**

➔ The information regarding the Size of the Pizza is available at **LS[7].** It is presumed that there are **04** Sizes of Pizza. **S** is **Small, R** is **Regular, M** is **Medium** and **L** is **Large.**

➔ The information on **ORDER_EVENT** is available at **LS[11].**

➔ If **SIZE** is **S** and **ORDER_EVENT** is **Delivered** then the output is written using a **Write** function. **Small_Pizza_Sold** is the **Key** and **1** is the **Value.**

➔ If **SIZE** is **R** and **ORDER_EVENT** is **Delivered** then the output is written using a **Write** function. **Regular_Pizza_Sold** is the **Key** and **1** is the **Value.**

➔ If **SIZE** is **M** and **ORDER_EVENT** is **Delivered** then the output is written using a **Write** function. **Medium_Pizza_Sold** is the **Key** and **1** is the **Value.**

➔ If **SIZE** is **L** and **ORDER_EVENT** is **Delivered** then the output is written using a **Write** function. **Large_Pizza_Sold** is the **Key** and **1** is the **Value.**

➔ It is presumed that the **ORDER_EVENT** could have multiple values like **Placed, Initiated, On The Way, Cancelled** And **Delivered.** The **Delivered** status will ensure that the Pizza is sold as a Delivered status cannot be changed to **Cancelled.** At all other stages, the **ORDER_EVENT** could be changed to **Cancelled.**

➔ The output of this **MAP** will be aggregated and provided as input to the **REDUCE.**

- For example, the output of the MAP would be (Small_Pizza_Sold, 1), (Small_Pizza_Sold, 1), (Regular_Pizza_Sold, 1), (Medium_Pizza_Sold, 1), (Large_Pizza_Sold, 1), (Large_Pizza_Sold, 1) .
- The aggregated result will be (Small_Pizza_Sold, {1,1}), (Regular_Pizza_Sold, {1}), (Medium_Pizza_Sold, {1}), (Large_Pizza_Sold, {1,1})

REDUCE_TOTAL_PIZZA_SOLD_CATEGORIZED_BY_SIZE (Key, ValueList)

```
START
   Count = 0
   FOR i = 1 To ValueList.length
      Count = Count + 1
   Write (Key, Count)
END
```

REDUCE DOCUMENTATION FOR QUESTION 4.II

➔ The input to the **REDUCE** would be the aggregated output of the **MAP.** For example, (Small_Pizza_Sold, {1,1}), (Regular_Pizza_Sold, {1}), (Medium_Pizza_Sold, {1}) and (Large_Pizza_Sold, {1,1})

➔ The **REDUCE** will loop through the ValueList and will count the total items in the list.

➔ The output of the **REDUCE** will be Key and Count. For example, (Small_Pizza_Sold, 2), (Regular_Pizza_Sold, 1), (Medium_Pizza_Sold, 1) and (Large_Pizza_Sold, 2).

III. Find out how many cheese burst pizzas were sold

```
MAP_TOTAL_CHEESE_BURST_PIZZA_SOLD (Key, Value)
START
   LS = Split (Value, “\t”)
   IS_CHEESE_BURST = LS[6]
   ORDER_EVENT = LS[11]
   IF ( IS_CHEESE_BURST == “Y” AND ORDER_EVENT == “Delivered”
      Write (Cheese_Burst_Pizza_Sold, 1)
END
```

MAP DOCUMENTATION FOR QUESTION 4.III

➔ A transaction record is stored in **Value.** The **Split** function extracts **12** values out of the record using a **tab** delimiter and stores these values in a **List LS.**

➔ The information whether a Pizza is **Cheese Burst or Not** is available at **LS[6].**

➔ The information on **ORDER_EVENT** is available at **LS[11].**

➔ If **IS_CHEESE_BURST** is **Y** and **ORDER_EVENT** is **Delivered** then the output is written using a **Write** function.
Cheese_Burst_Pizza_Sold is the **Key** and **1** is the **Value.**

➔ It is presumed that the **ORDER_EVENT** could have multiple values like **Placed, Initiated, On The Way, Cancelled** And **Delivered.** The **Delivered** status will ensure that the Pizza is sold as a Delivered status cannot be changed to **Cancelled.** At all other stages, the **ORDER_EVENT** could be changed to **Cancelled.**

➔ The output of this **MAP** will be aggregated and provided as input to the **REDUCE.**

- For example, the output of the MAP would be (Cheese_Burst_Pizza_Sold, 1), (Cheese_Burst_Pizza_Sold, 1).
- The aggregated result will be (Cheese_Burst_Pizza_Sold, {1,1})

REDUCE_TOTAL_CHEESE_BURST_PIZZA_SOLD (Key, ValueList)

```
START
   Count = 0
   FOR i = 1 To ValueList.length
      Count = Count + 1
   Write (Key, Count)
END
```

REDUCE DOCUMENTATION FOR QUESTION 4.III

➔ The input to the **REDUCE** would be the aggregated output of the **MAP.** For example, (Cheese_Burst_Pizza_Sold, {1,1})

➔ The **REDUCE** will loop through the ValueList and will count the total items in the list.

➔ The output of the **REDUCE** will be Key and Count. For example, (Cheese_Burst_Pizza_Sold, 2).

IV. Find out how many small cheese burst pizzas were sold. Ideally, the count should be 0 because cheese burst is available for medium and large

```
MAP_TOTAL_SMALL_CHEESE_BURST_PIZZA_SOLD (Key, Value)
START
  LS = Split (Value, “\t”)
  IS_CHEESE_BURST = LS[6]
  SIZE = LS[7]
  ORDER_EVENT = LS[11]
  IF ( IS_CHEESE_BURST == “Y”  AND  SIZE == “S” AND
  ORDER_EVENT== ”Delivered” )
    Write (Small_Cheese_Burst_Pizza_Sold, 1)
END
```

MAP DOCUMENTATION FOR QUESTION 4.IV

➔ A transaction record is stored in **Value.** The **Split** function extracts **12** values out of the record using a **tab** delimiter and stores these values in a **List LS.**

➔ The information whether a Pizza is **Cheese Burst or Not** is available at **LS[6].**

➔ The information regarding the Size of the Pizza is available at **LS[7].** It is presumed that there are **04** Sizes of Pizza. **S** is **Small, R** is **Regular, M** is **Medium** and **L** is **Large.**

➔ The information on **ORDER_EVENT** is available at **LS[11].**

➔ If **IS_CHEESE_BURST** is **Y, SIZE** is **S** and **ORDER_EVENT** is **Delivered** then the output is written using a **Write** function. **Small_Cheese_Burst_Pizza_Sold** is the **Key** and **1** is the **Value.**

➔ It is presumed that the **ORDER_EVENT** could have multiple values like **Placed, Initiated, On The Way, Cancelled** And **Delivered.** The **Delivered** status will ensure that the Pizza is sold as a Delivered status cannot be changed to **Cancelled.** At all other stages, the **ORDER_EVENT** could be changed to **Cancelled.**

➔ The output of this **MAP** will be aggregated and provided as input to the **REDUCE.**

- For example, the output of the MAP would be (Small_Cheese_Burst_Pizza_Sold, 1), (Small_Cheese_Burst_Pizza_Sold, 1).
- The aggregated result will be (Small_Cheese_Burst_Pizza_Sold, {1,1})

REDUCE_ TOTAL_SMALL_CHEESE_BURST_PIZZA_SOLD (Key, ValueList)

```
START
   Count = 0
   FOR i = 1 To ValueList.length
      Count = Count + 1
   Write (Key, Count)
END
```

REDUCE DOCUMENTATION FOR QUESTION 4.IV

➔ The input to the **REDUCE** would be the aggregated output of the **MAP**. For example, (Small_Cheese_Burst_Pizza_Sold, {1,1})

➔ The **REDUCE** will loop through the ValueList and will count the total items in the list.

➔ The output of the **REDUCE** will be Key and Count. For example, (Small_Cheese_Burst_Pizza_Sold, 2).

V. Find out the number of cheese burst pizzas whose cost is below Rs 500

```
MAP_TOTAL_CHEESE_BURST_PIZZA_LESS_THAN_RS_500 (Key, Value)
START
   LS = Split (Value, “\t”)
   IS_CHEESE_BURST = LS[6]
   PRICE = LS[9]
   IF ( IS_CHEESE_BURST == “Y”  AND  PRICE  < 500)
      Write (Cheese_Burst_Pizza_Less_Than_500, 1)
END
```

MAP DOCUMENTATION FOR QUESTION 4.V

➔ A transaction record is stored in **Value.** The **Split** function extracts **12** values out of the record using a **tab** delimiter and stores these values in a **List LS.**

➔ The information whether a Pizza is **Cheese Burst or Not** is available at **LS[6].**

➔ The information regarding the **Price** of a Pizza is available at **LS[9].**

➔ If **IS_CHEESE_BURST** is **Y** and **PRICE** is less than **500** then the output is written using a **Write** function.
Cheese_Burst_Pizza_Less_Than_500 is the **Key** and **1** is the **Value.**

➔ Unlike other algorithms, we have not considered the **ORDER_EVENT** as the question does not specifically talk about **Sold Pizza.**

➔ The output of this **MAP** will be aggregated and provided as input to the **REDUCE.**

- For example, the output of the MAP would be
  (Cheese_Burst_Pizza_Less_Than_500, 1),
  (Cheese_Burst_Pizza_Less_Than_500, 1).
- The aggregated result will be
  (Cheese_Burst_Pizza_Less_Than_500, {1,1})

REDUCE_TOTAL_CHEESE_BURST_PIZZA_LESS_THAN_RS_500 (Key, ValueList)

```
START
   Count = 0
   FOR i = 1 To ValueList.length
      Count = Count + 1
   Write (Key, Count)
END
```

REDUCE DOCUMENTATION FOR QUESTION 4.V

➔ The input to the **REDUCE** would be the aggregated output of the **MAP**. For example, (Cheese_Burst_Pizza_Less_Than_500, {1,1})

➔ The **REDUCE** will loop through the ValueList and will count the total items in the list.

➔ The output of the **REDUCE** will be Key and Count. For example, (Cheese_Burst_Pizza_Less_Than_500, 2).