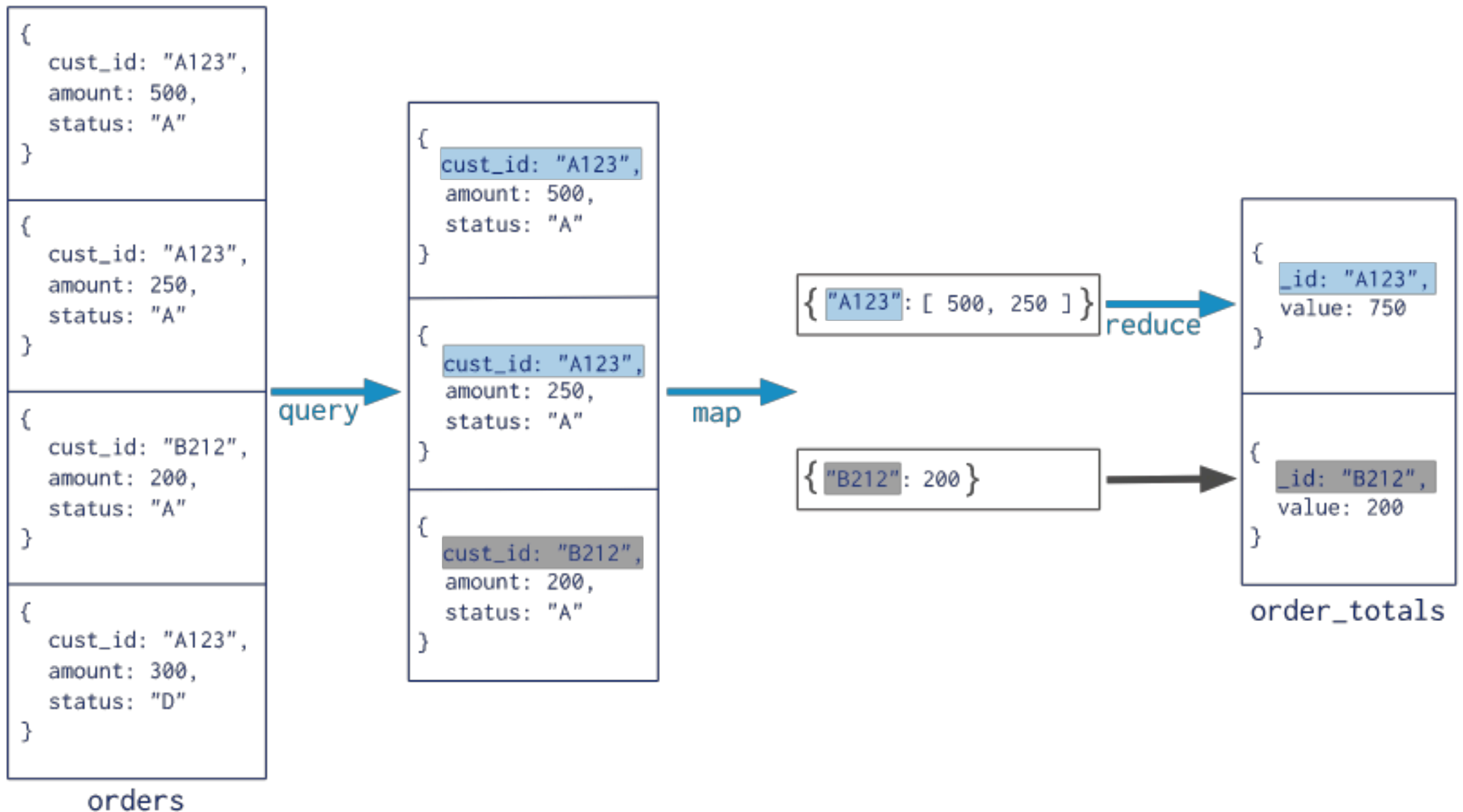Collection

```
db.orders.mapReduce(
        map      ⟶  function() { emit( this.cust_id, this.amount ); },
        reduce   ⟶  function(key, values) { return Array.sum( values ) },
                    {
        query    ⟶    query: { status: "A" },
        output   ⟶    out: "order_totals"
                    }
                )
```

orders

```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
{
  cust_id: "A123",
  amount: 300,
  status: "D"
}
```

query ⟶

```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
```

map ⟶

```
{ "A123": [ 500, 250 ] }
```
reduce ⟶

```
{ "B212": 200 }
```

order_totals

```
{
  _id: "A123",
  value: 750
}
{
  _id: "B212",
  value: 200
}
```

This map-reduce operation applies the **map** phase to documents in the collection that match some query condition (i.e,. customer IDs).
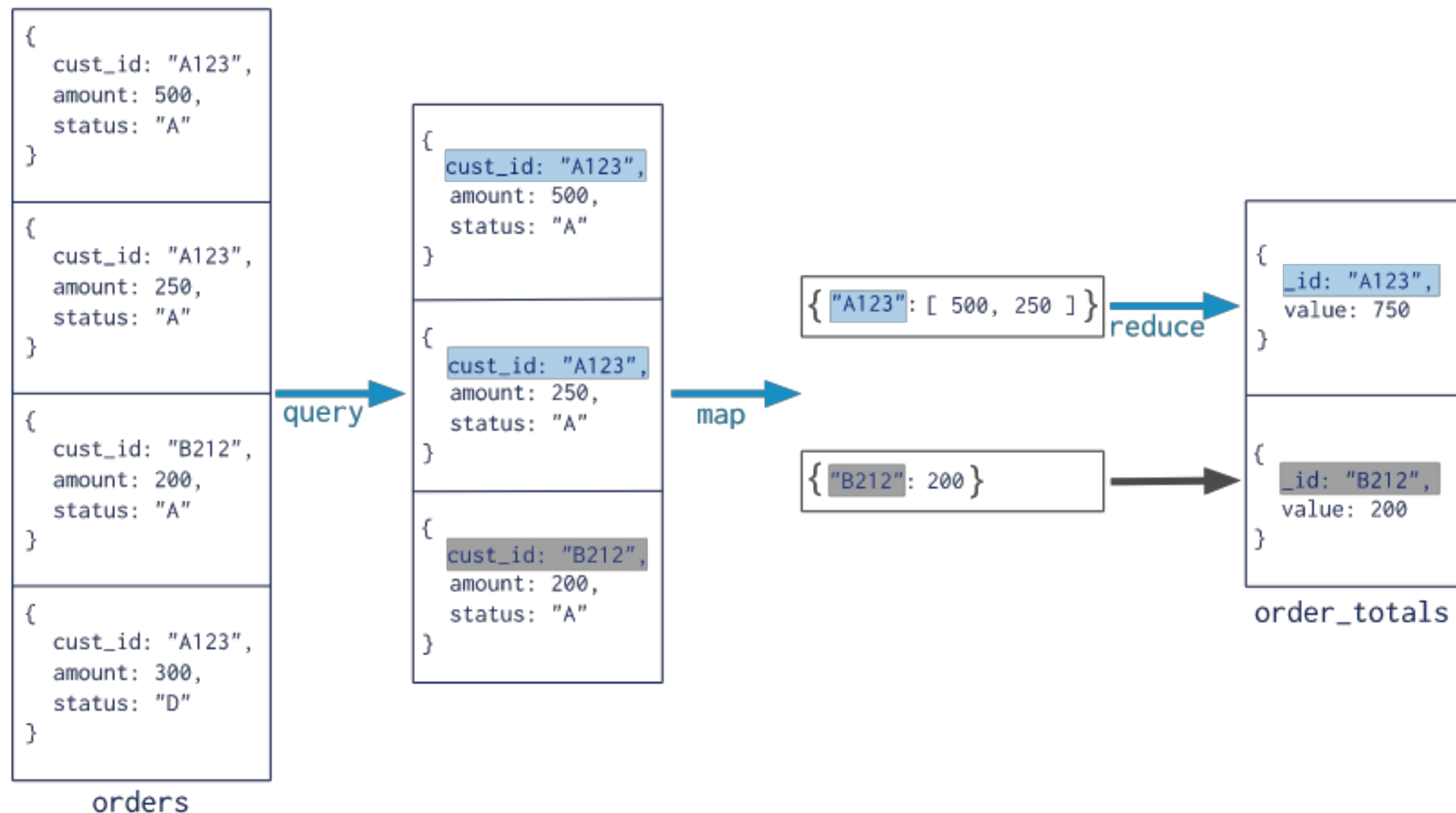
For those keys (customer IDs) with multiple values (i.e., orders), the **reduce** phase collects and condenses the aggregated data.

Finally, the results are aggregated and stored in some collection.

This map-reduce operation applies the **map** phase to documents in the collection that match some query condition (i.e,. customer IDs).

For those keys (customer IDs) with multiple values (i.e., orders), the **reduce** phase collects and condenses the aggregated data.

Finally, the results are aggregated and stored in some collection.

```
{
    cust_id: "A123",
    amount: 500,
    status: "A"
}

{
    cust_id: "A123",
    amount: 250,
    status: "A"
}

{
    cust_id: "B212",
    amount: 200,
    status: "A"
}

{
    cust_id: "A123",
    amount: 300,
    status: "D"
}
```

orders

query →

```
{
    cust_id: "A123",
    amount: 500,
    status: "A"
}

{
    cust_id: "A123",
    amount: 250,
    status: "A"
}

{
    cust_id: "B212",
    amount: 200,
    status: "A"
}
```

map →

```
{ "A123": [ 500, 250 ] }
```

reduce →

```
{ "B212": 200 }
```

```
{
    _id: "A123",
    value: 750
}

{
    _id: "B212",
    value: 200
}
```

order_totals

BY THE END OF THIS SESSION YOU SHOULD BE ABLE TO:

- Conceptual:

  - Identify the 4Vs of Big Data, defining each criterion.

  - Understand the difference between structured and unstructured data.

  - Differentiate between serial and parallel processing.

  - Explain parallelization in terms of its advantages in dealing with Big Data.

  - Identify the map and reduce phases in MapReduce and explain what is happening in both parts.