Aggregations MongoDB

...

~Hardeep Singh

Some Queries and Options in Aggregation

Queries - $match, $sort, $skip, $limit, $group,

$project, $redact, $unwind, $out, $sample,

$geoNear, $lookup, $indexState

Options - Explain, allowDiskUse, cursor, readConcern,

bypassDocumentValidation

Single Purpose Operations -

- **EstimatedDocumentCount** - Helps to fetch total count in collections.

  Example - `db.orders.estimatedDocumentCount( {})`

- Count - Helps to fetch total count in collections, you can also provide query and options in it.

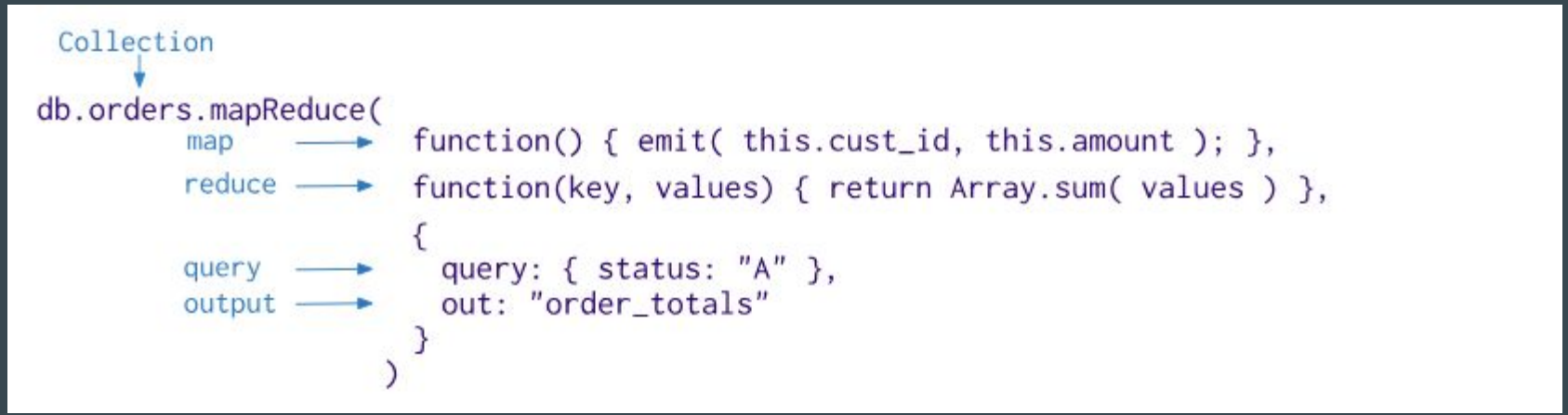
  Example - `db.orders.count({ query, options })`

- Distinct - Helps to fetch Array of distinct count in collection.

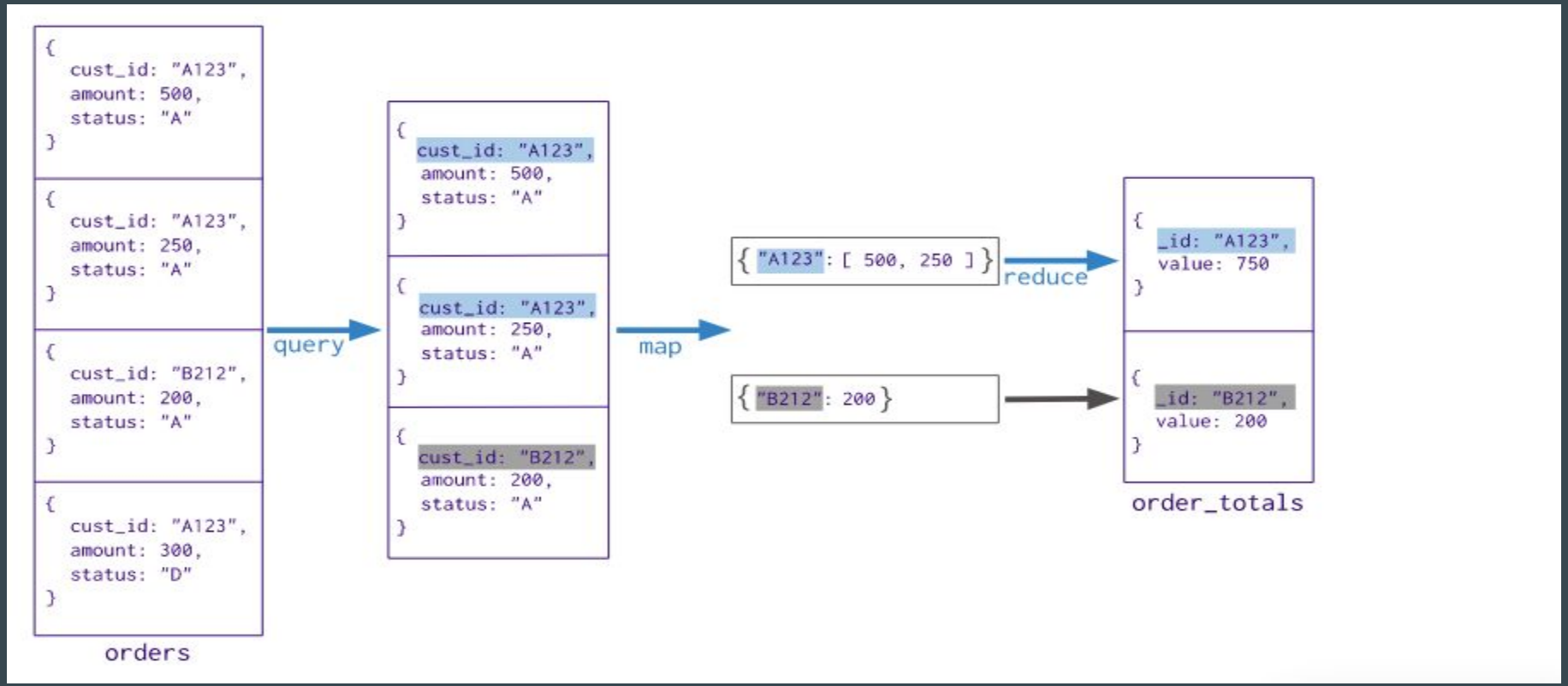
  Example - `db.orders.distinct({ field: string,query,options })`

Map-Reduce Function

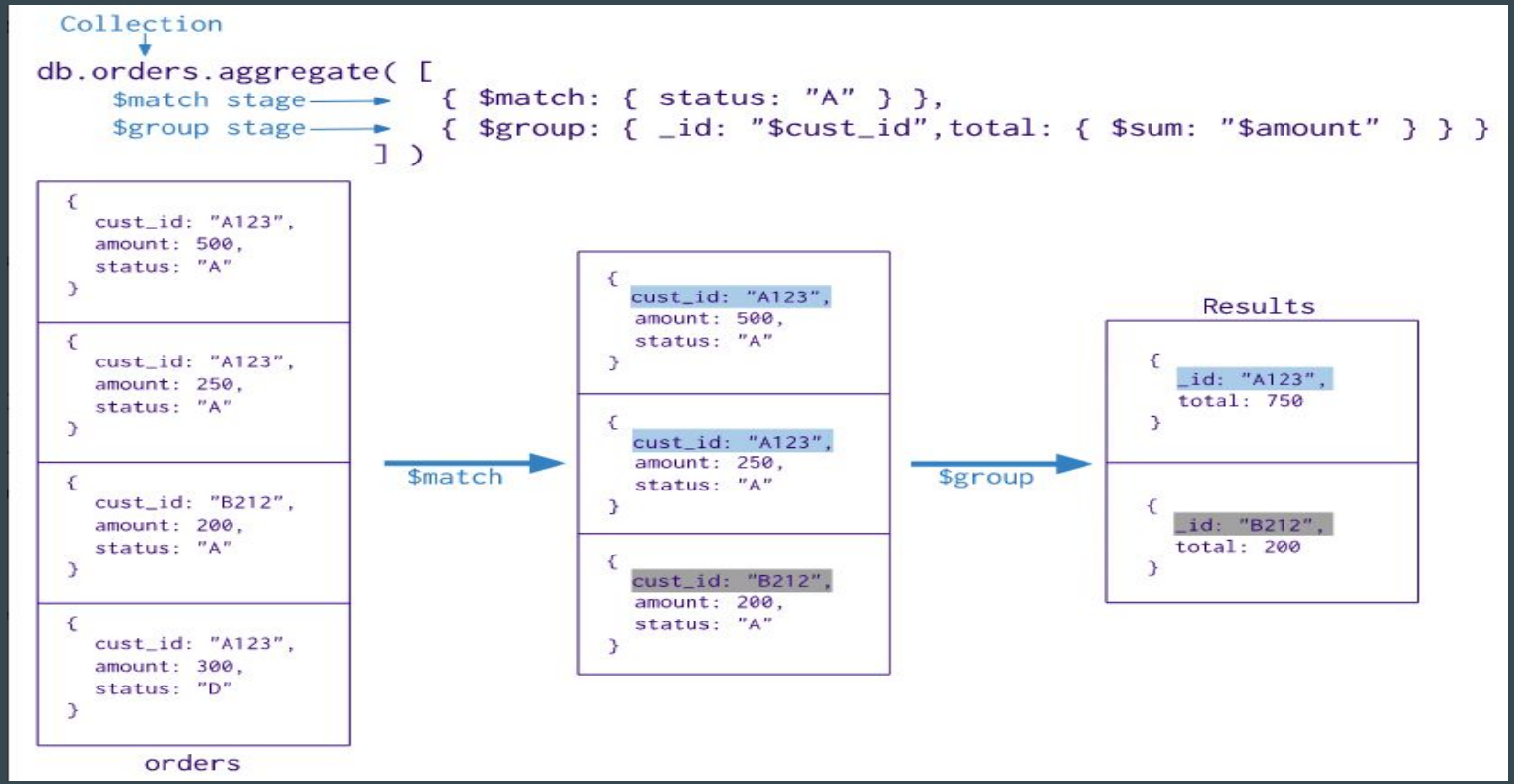
The Query, we perform here is of 3 parameters, 1st param is map function, 2nd param is reduce function, 3rd param is object (having two key value pairs.). Total funciton is doing 4 process on single query.



Result Map-Reduce Function



Aggregation Chain Mechanism



Aggregations Queries (Not Understand By Name)

- **$project** - The `$project` takes a document that can specify the inclusion of fields, the suppression of the `_id` field, the addition of new fields, and the resetting of the values of existing fields. Alternatively, you may specify the *exclusion* of fields.

```
{ $project: { item:1, element:0  } }
```

- **$unwind** - Open up the Array data into multiple documents from one documents.

```
db.inventory.aggregate(  [ { $unwind : "$sizes"  } ] )
```

- **$redact** - Reshapes each document in the stream by restricting the content for each document based on information stored in the documents themselves

- **$project** : Helps to joining tables with helps of ObjectId reference.

- **$geoNear:** Helps to fetch data on case of location in aggregation.

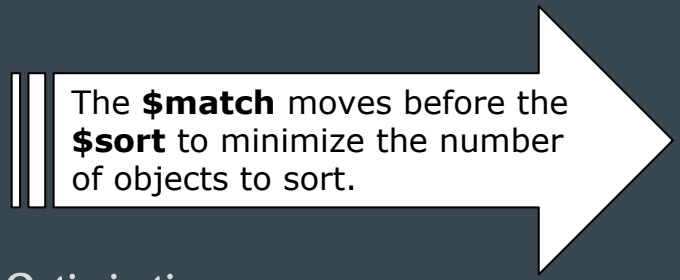
- **$sample**: Randomly selects the specified number of documents from its input.

- **$out**: The `$out` operation creates a new collection in the current database if one does not already exist. The collection is not visible until the aggregation completes. If the aggregation fails, MongoDB does not create the collection.

Aggregation Pipeline Optimization

- **$sort + $match** Sequence Optimization

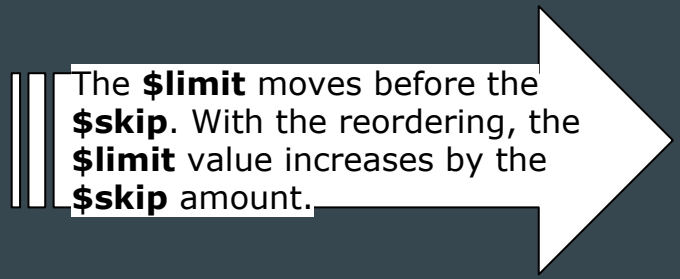
```
{ $sort: { age : -1 } },
{ $match: { status: 'A' } }
```




```
{ $match: { status: 'A' } }
{ $sort: { age : -1 } }
```

- **$skip + $limit** Sequence Optimization

```
{ $skip: 10 },
{ $limit: 5 }
```




```
{ $limit: 15 },
{ $skip: 10 }
```

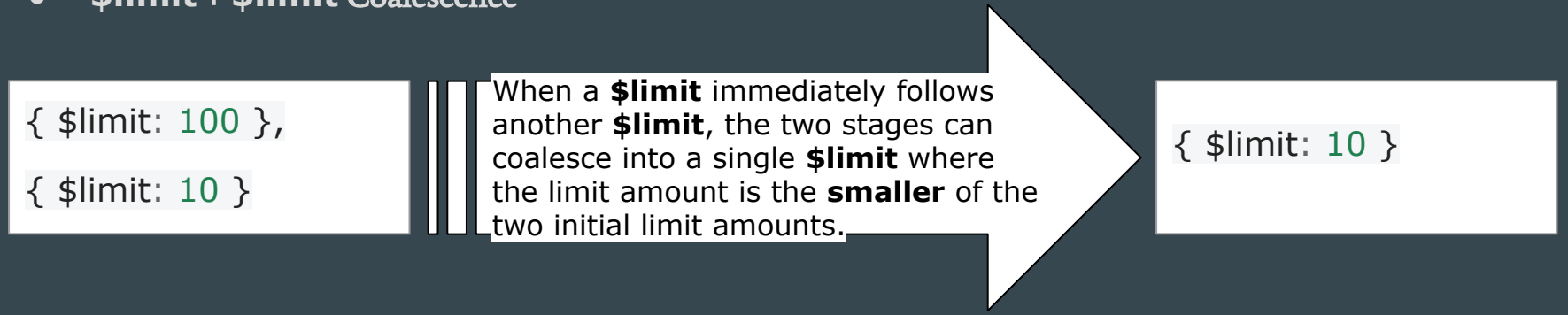
- **$project + $skip or $limit Sequence Optimization**

```
{ $sort: { age : -1 } },
{ $project: { status: 1, name: 1
} },
{ $limit: 5 }
```

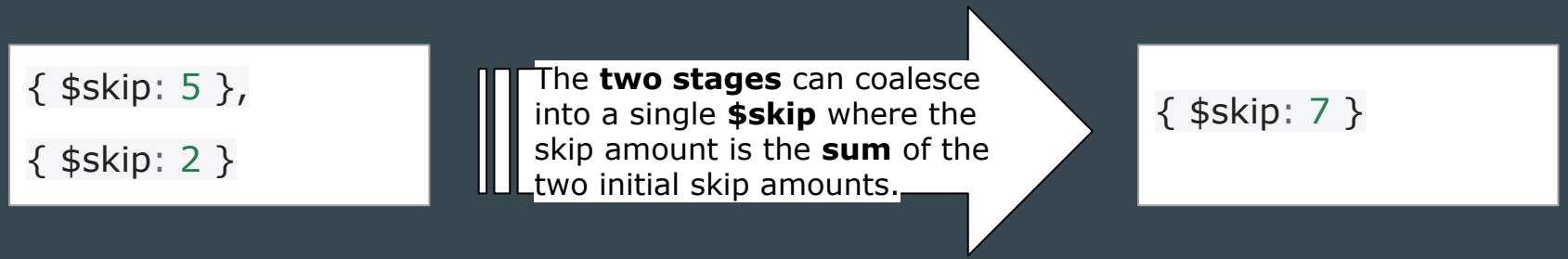
When you have a sequence with **$project** followed by either **$skip** or **$limit**, the **$skip** or **$limit** moves before **$project**.

```
{ $sort: { age : -1 } },
{ $limit: 5 }
{ $project: { status: 1, name: 1
} },
```

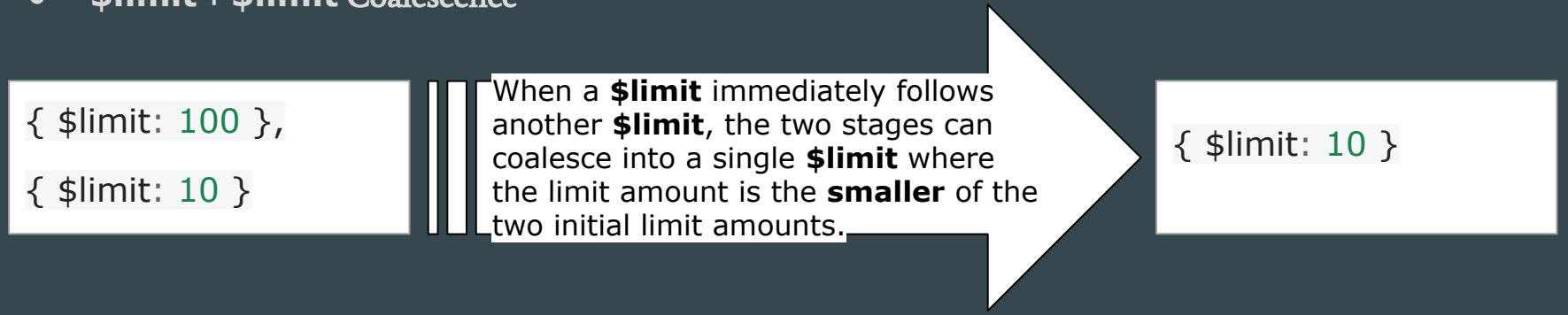
- **$limit + $limit** Coalescence



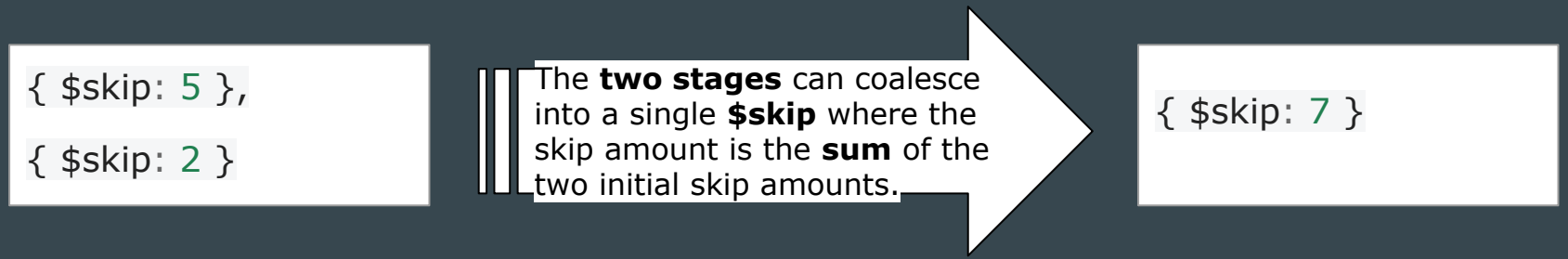
- **$skip + $skip** Coalescence



- **$limit + $limit** Coalescence



- **$skip + $skip** Coalescence



- $lookup + $unwind Coalescence ($unwing, **by default preserveNullAndEmptyArrays is true** (show all the Nuall or Empty things))

```
{  $lookup: {
    from: "otherCollection",
    as: "resultingArray",
    localField: "x",
    foreignField: "y"
  }},
{ $unwind: "$resultingArray"}
```

The **$unwind** operates on the **as** field of the **$lookup**, the optimizer can coalesce the **$unwind** into the **$lookup** stage. This avoids creating large intermediate documents.

```
{
   $lookup: {
       from: "otherCollection",
       as: "resultingArray",
       localField: "x",
       foreignField: "y",
       unwinding: {

preserveNullAndEmptyArrays: false
      }
   }
}
```

Any Questions ?

Questions -

1. Add 50 User and 100 Address Collection, One User has multiple Address.

User properties - name, age, companyName, type, fatherName, motherName, Address ArrayIds as references.

Address Properties - houseNo, landmark, city, state, pincode.

Write a query for to fetch all address data from address in users. Limit should be 10 and skiping first five.

2. Fetch & Sort User data on base of user names, Sort Address cities in User table. Limit should be 5 and skipping first 5.

3. Fetch only address selective data (UserID:””, AddressData:””) from user table, keys should be like above.

4. Create indexes in Address table, on city and state. And Sort it.

Thank you

~ Hardeep Singh