Aggregation Pipeline Stages

We have below Stages (Its not a complete list,For complete list visit [MongoDB official Site](https://docs.mongodb.com/v3.0/reference/operator/aggregation/)) in Aggregation Pipeline and Mapping with Sql Server so that we can have a clear picture assuming that we have a little knowledge of SQL server

| **In SQL** | **In MongoDB** | **Description** |
| --- | --- | --- |
| Select | $project | Passes the fields to next stage with existing Fields or with New fields.We can add new Fields dynamically |
| Where | $match | This will filter the documents and will pass only matching documents to next pipeline stage. |
| Limit | $limit | limit the first x unmodified documents and pass them to next stage of pipeline. x is the number of the documents which will pass through the next stage of pipeline. |
| GroupBy | $group | This will group the documents and pass them to the next stage of Aggregation pipeline. |
| OrderBy | $sort | It will change the order of documents either in ascending or descending. |
| Sum | $sum | To calculate the sum of all the numeric values. |
| Join | $lookup | It will perform the left outer join with another collection in same database. |

So much talk Its time to understand some pipeline stages and Operators with some examples.

Now suppose we have a School database and have a Student Collection as below

Hide   Copy Code

db.Student.insert({StudentName : "Vijay",Section : "A",Marks:70,Subject:["Hindi","English","Math"]})

db.Student.insert({StudentName : "Gaurav",Section : "A",Marks:90,Subject:["English"]})

db.Student.insert({StudentName : "Ajay",Section : "A",Marks:70,Subject:["Math"]})

db.Student.insert({StudentName : "Ankur",Section : "B",Marks:10,Subject:["Hindi"]})

db.Student.insert({StudentName : "Sunil",Section : "B",Marks:70,Subject:["Math"]})

db.Student.insert({StudentName : "Preeti",Section : "C",Marks:80,Subject:["Hindi","English"]})

db.Student.insert({StudentName : "Anuj",Section : "C",Marks:50,Subject:["English"]})

db.Student.insert({StudentName : "Palka",Section : "D",Marks:40,Subject:["Math"]})

db.Student.insert({StudentName : "Soniya",Section : "D",Marks:20,Subject:["English","Math"]})

We will see different stages and how they works on this Student Collection.So Let's ready for some good stuff

**$match**

**$match** is similar to **Where**in SQL. In SQL we use Where to filter the data and same is here.If we need to pass only a subset of our data in next stage of Aggregation Pipeline then we use **$match**.**$match**filters the data and pass the matching data to the next stage of Pipeline.

**Example 1 :**Suppose we want to filter data based on **Section A** in Student Collection then we will use **$match**as below

Hide   Copy Code

db.Student.aggregate

(

[

{

"$match":

{

"Section":"A"

}

}

]

)

This will filter the data according to our $match and will pass only 3 rows to next Stage of pipeline where **Section is A.**

**Result**



**Example 2 :**Suppose if want to find out all the records where **Section is A** and **Marks**is greater then **80**

Hide   Copy Code

db.Student.aggregate (

[

   {

       $match:

       {

           $and:[{Section:'A'},{Marks: {"$gt" :80}}]

       }

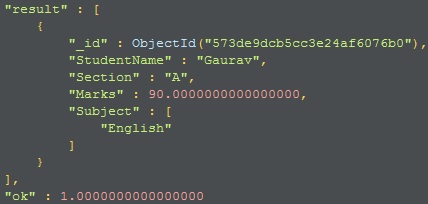
    }

 ]

)

This will give us one record

**Result**



**NOTE : There can be more than one $match in Aggregate Function.**

**$project** :

We can compare this clause with **SELECT**in SQL. We can select certain fields, rename Fields from documents though **$project**. In short **$project** reshape the documents by adding/removing or renaming the documents for the next stage of pipeline. In $project we use 1 or true if we want to include the Field and 0 or false if we want to exclude a particular field.

**Example 1 :** In the below query we want only StudentName,Section and Marks from student collection then we will use the below query

Hide   Copy Code

db.Student.aggregate

(

 [

  {

       "$project":{StudentName : 1,Section:1,Marks:1}

  }

 ]

)

**Example 2 :** Now if we want to find out **StudentName**,**Section**and **Marks**from Student Collection where **Section**is **'A'** than we will use $**project**and $**match**both

Hide   Copy Code

db.Student.aggregate

(

 [

  {

     "$match":

     {

        "Section":"A"

     }

  },

  {

       "$project":

     {

         StudentName : 1,Section:1,Marks:1

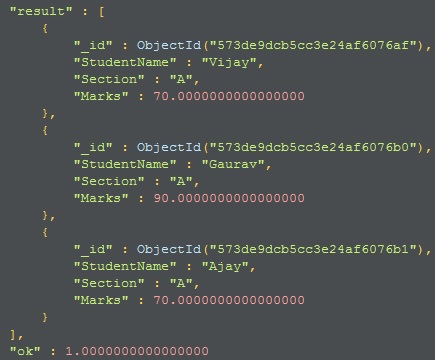
     }

  }

 ]

)

**Result**



**NOTE : \_id will be visible by default, if we don't want the \_id field in result then we need to remove it explicitly as below**

Hide   Copy Code

"$project":{StudentName : 1,Section:1,Marks:1,\_id:0}

**$unwind**

**$unwind**works on the the array field inside the documents. **$unwind**creates a new document for each array element in an array.**$unwind**output is a new document of each entry of an array inside a document.we use **$unwind** to flattens the data.

**Example 1 :** Suppose we want to apply $unwind on a document where name is **Vijay**.In this document we have an array field named **Subject**which contains three subjects named **Hindi**,**English**and **Math**. Let's see what $unwind will do with this document

Hide   Copy Code

db.Student.aggregate

(

 [

  {

     "$match":

     {

        "StudentName":"Vijay"

     }

  },

  {

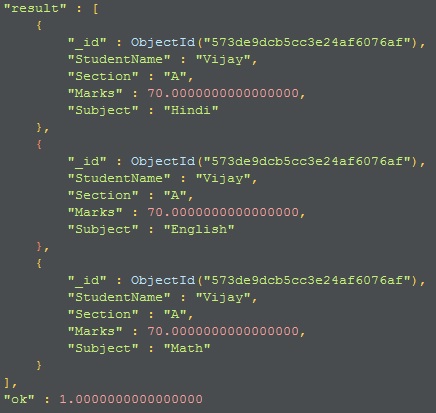
       "$unwind":"$Subject"

  }

 ]

)

**Result**



**Example 2 :**   If we want to select only **StudentName**,**Section**,**Marks**,**Subject**then we can use**$project**along with**$match and $unwind**as below

Hide   Copy Code

db.Student.aggregate

(

 [

  {

     "$match":

      {

        "StudentName":"Vijay"

      }

  },

  {

       "$unwind":"$Subject"

  },

{

       "$project":

      {

         StudentName : 1,Section:1,Marks:1,Subject:1

      }

  }

 ]

)

MongoDB is an Schema less so it might be possible that some documents does not contain array or some contains empty array so will **$unwind** will work of such documents? answer is yes. after MongoDB release 3.2 if document contains empty array or does not contain array then pipeline will ignore the input document and will not generate output document for such document.

before MongoDB release 3.2 if we don't have an array or we have a empty array and we are using $unwind then MongoDB generates an error.

**Example 3 :** Let me add two documents as below in our Student Collection, In first document we have an empty array and in second document we don't have any array field.

Hide   Copy Code

db.Student.insert({StudentName : "Tarun",Section : "A",Marks:95,Subject:[]})

db.Student.insert({StudentName : "Saurabh",Section : "A",Marks:95})

Now let me run the $unwind again for the document where StudentName is Tarun and Saurabh

Hide   Copy Code

db.Student.aggregate

(

 [

  {

     "$match":

     {

        "StudentName":{$in:["Saurabh","Tarun"]}

     }

  },

  {

       "$unwind":"$Subject"

  }

 ]

)

So above query will not generate any output document because  array is missing or empty.

includeArrayIndex parameter

**Example 4 :** In $unwind we can pass second parameter named includeArrayIndex which we can pass in $unwind if want to include ArrayIndex in result.

Hide   Copy Code

db.Student.aggregate

(

 [

  {

     "$match":

     {

        "StudentName":"Vijay"

     }

  },

  {

       "$unwind":{ path: "$Subject", includeArrayIndex: "arrayIndex" }

  }

 ]

)

**Result**



**$group**

MongoDB use **$group** to group the documents by some specified expression.$group is similar to Group clause in SQL. Group in SQL is not possible without any Aggregate Function and the same is here. We can not group in MongoDB without Aggregate Functions. let's understand with an example

**Example 1** : Suppose we want to find out Total Marks group by Section then we will use $group as below

Hide   Copy Code

db.Student.aggregate ([

   {

      "$group":

      {

         "\_id":

         {

            "Section" : "$Section"

         },

         "TotalMarks":

         {

            "$sum": "$Marks"

         }

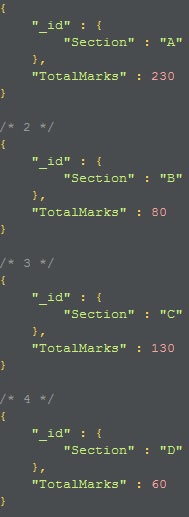
      }

   }

])

In this query **\_id**Field is mandatory. In **\_id** we pass the field on which we want to group the documents.This will give us below result

**Result**



**Example 2 :**If we want to fetch Total Marks for only Section 'A' then we can pass a **$match** also.

Hide   Copy Code

db.Student.aggregate ([

   {

       "$match":{Section :'A'}

   },

   {

      "$group":

      {

         "\_id":

         {

            "Section" : "$Section"

         },

         "TotalMarks":

         {

            "$sum": "$Marks"

         }

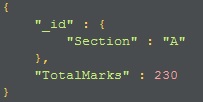
      }

   }

])

This will Sum the Total Marks of Section 'A' only.

**Result**



**Example 3** :

Suppose we want to fetch the count of students in each section and Total marks and average marks as well

Hide   Copy Code

db.Student.aggregate ([

   {

      "$group":

      {

         "\_id":

         {

            "Section" : "$Section"

         },

         "TotalMarks":

         {

            "$sum": "$Marks"

         },

         "Count":{ "$sum" : 1},

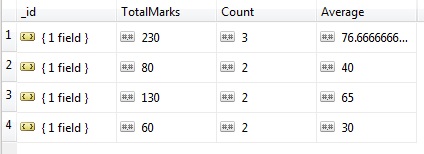
         "Average" : {"$avg" : "$Marks"}

      }

   }

])

**Result**



**Example 4 :**If we want to rename the column Names in above query(Section to SectionName and TotalMarls to Total)  then we can use **$project** along with **$group** as below

Hide   Copy Code

db.Student.aggregate ([

   {

      "$group":

      {

         "\_id":

         {

            "Section" : "$Section"

         },

         "TotalMarks":

         {

            "$sum": "$Marks"

         },

         "Count":{ "$sum" : 1},

         "Average" : {"$avg" : "$Marks"}

      }

   },

   {

       "$project" :

       {

           "SectionName" : "$\_id.Section",

           "Total" : "$TotalMarks"

       }

   }

])

**$sort**

**$sort** is similar to  orderby clause in SQL server. In MongoDB we have **$sort** for this. **$sort** will sort the documents in either ascending or descending order as below. MongoDB uses 1 for ascending and -1 for descending

**Example 1 :**If we want to  sort the result in descending order by SectionName then we can use $sort

Hide   Copy Code

db.Student.aggregate ([

   {

      "$group":

      {

         "\_id":

         {

            "Section" : "$Section"

         },

         "TotalMarks":

         {

            "$sum": "$Marks"

         },

         "Count":{ "$sum" : 1},

         "Average" : {"$avg" : "$Marks"}

      }

   },

   {

       "$project" :

       {

           "SectionName" : "$\_id.Section",

           "Total" : "$TotalMarks"

       }

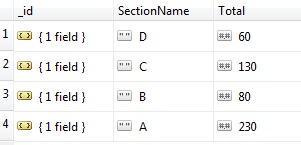
   },

   {

       "$sort":{"SectionName":-1}

   }

])



**$limit**

**$limit** operator use to pass n documents to next pipe line stage where n is the limit.n is the number of documents

**Example 1 :** If we want to sort the documents as in above query and we need to pass only 2 documents to the next stage of pipeline then we use $limit .

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

db.Student.aggregate ([

   {

      "$group":

      {

         "\_id":

         {

            "Section" : "$Section"

         },

         "TotalMarks":

         {

            "$sum": "$Marks"

         },

         "Count":{ "$sum" : 1},

         "Average" : {"$avg" : "$Marks"}

      }

   },

   {

       "$project" :

       {

           "SectionName" : "$\_id.Section",

           "Total" : "$TotalMarks"

       }

   },

   {

       "$sort":{"SectionName":-1}

   },

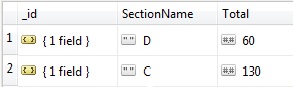
   {

       "$limit" : 2

   }

])

**Result**



**$skip**

$skip is use to skip first n documents and remaining will be passed in next pipeline.n is the number of documents which we want to skip

**Example 1 :**In the above example if we want to skip first 1 document and then we want to pass the next 2 documents to the next stage of pipeline then we  will use the below query

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

db.Student.aggregate ([

   {

      "$group":

      {

         "\_id":

         {

            "Section" : "$Section"

         },

         "TotalMarks":

         {

            "$sum": "$Marks"

         },

         "Count":{ "$sum" : 1},

         "Average" : {"$avg" : "$Marks"}

      }

   },

   {

       "$project" :

       {

           "SectionName" : "$\_id.Section",

           "Total" : "$TotalMarks"

       }

   },

   {

       "$sort":{"SectionName":-1}

   },

   {

       "$skip" : 1

   },

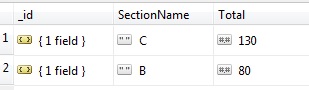
   {

       "$limit" : 2

   }

])

**Result**



**$lookup**

This is the most awaited feature in MongoDB. $lookup is equal to joins in SQL.$lookup comes with MongoDB release 3.2. Before Version MongoDB version 3.2 there were no concept of joins (In my first Articles I mentioned that MongoDB does not support Join in my first Article). Let's Understand this by using an example

**Example 1 :**Suppose we have two collection named Country and City as below

Hide   Copy Code

db.Country.insert({"\_id":1,"Name":"India"})

db.Country.insert({"\_id":2,"Name":"US"})

db.Country.insert({"\_id":3,"Name":"UK"})

db.Country.insert({"\_id":4,"Name":"Australia"})

db.City.insert({"\_id":1,"Name":"Delhi","CountryID":1})

db.City.insert({"\_id":2,"Name":"Noida","CountryID":1})

db.City.insert({"\_id":3,"Name":"Chicago","CountryID":2})

db.City.insert({"\_id":4,"Name":"London","CountryID":3})

db.City.insert({"\_id":5,"Name":"Bristol","CountryID":3})

db.City.insert({"\_id":6,"Name":"Sydney","CountryID":4})

If we want to fetch all the cities associated with countries then we will use **$lookup** as below

Hide   Copy Code

db.City.aggregate([

    {

      $lookup:

        {

          from: "Country",

          localField: "CountryID",

          foreignField: "\_id",

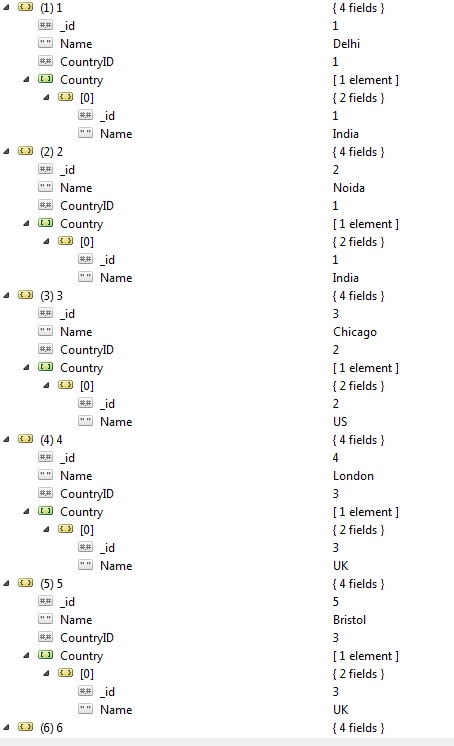
          as: "Country"

        }

   }

])

In the Above query we are joining City with Country collection, CountryID is local field of City and \_id is a foreign field of Country.



**$redact**

MongoDB uses the $redact to restrict the content of the documents based on the information stored in the document itself.to understand this better i will cover first $cond,$setIntersection,$size before $redact.

$cond :-

$cond checks a boolean expression and return expressions according to result. This is not a Stage in pipeline but good to know how $cond works because we are going to use it Shortly.

$cond follow the syntax as below

Hide   Copy Code

{ $cond: { if: (boolean-expression), then: (true-case), else: (false-case) } }

**Example 1 :**In our Student Collection If we want to display Good in result if marks is greater then 70 and Poor if marks is less then 70 then we can use $cond as below

Hide   Copy Code

db.Student.aggregate(

[

{

$project:

{

StudentName: 1,

Result:

{

$cond: { if: { $gte: [ "$Marks", 70 ] }, then: "Good", else: "Poor" }

}

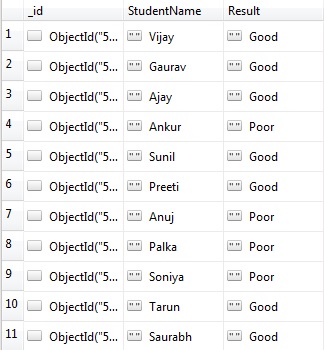
}

}

]

)

**Result**



$setIntersection :-

**$setIntersection**Takes 2 arrays as a Input and returns an array with the common element in both the array.

Suppose i have two arrays in a document in my Test collection as below

Hide   Copy Code

db.Test.insert({"Array1":["1","2","3"],"Array2":["1","2","3","4","5"]})

**Example :**If we want to find out the common elements between two arrays then we will use **$setIntersection** as below

Hide   Copy Code

db.Test.aggregate(

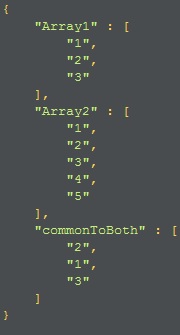
   [

     { $project: { "Array1": 1, "Array2": 1, commonToBoth: { $setIntersection: [ "$Array1", "$Array2" ] }, \_id: 0 } }

   ]

)

**Result :**



$size

$size counts and returns the total the number of items in an array.In the below query we are counting the element of Array1 and Array2

Hide   Copy Code

db.Test.aggregate(

   [

      {

         $project: {

            Array1count: { $size: "$Array1" },

            Array2count: { $size: "$Array2" }

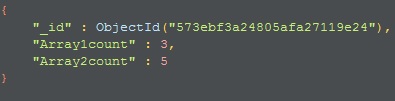
         }

      }

   ]

)

**Result**



So we are good with $cond,$setIntersection and $size and now let's understand $redant with an example but remove records where array is null or missing otherwise $redant will throw an exception. So i am removing two documents from Student collection where array is null or empty (both the documents where array is empty or missing marks is 95)

Hide   Copy Code

db.Student.remove({Marks:95})

Hide   Copy Code

var SubjectAccess=["Math","Hindi"];

db.Student.aggregate(

[{

    "$match": {"Section":"A"}

},

{

    $redact:{

         $cond: {

           if: { $gt: [ { $size: { $setIntersection: [ "$Subject", SubjectAccess ]{} }, 0 ] },

           then: "$$DESCEND",

           else: "$$PRUNE"

         }

        }

}])

Above query will check if Subject contains data either Hindi or Math then it will allow to pass the document to next stage of pipeline and it will restrict all the documents where subject does not contain either Math or Hindi and of course it will match the condition where section is "A"

**Result**



So we are done enough for day 4. Truly speaking its more than enough for day in fact for a week.