Agregação com o conjunto de dados de CEP

Nesta página

- Modelo de dados
- `aggregate()` Método
- Estados de retorno com populações acima de 10 milhões
- População média da cidade de retorno por estado
- Retornar cidades maiores e menores por estado

Os exemplos neste documento usam a `zipcodes`coleção. Esta coleção está disponível em: media.mongodb.org/zips.json . Use `mongoimport`para carregar esse conjunto de dados em sua `mongod`instância.

Modelo de dados

Cada documento da `zipcodes`coleção tem o seguinte formato:

```
{
  "_id" :  "10280" ,
  "city" :  "NEW YORK" ,
  "state" :  "NY" ,
  "pop" :  5574 ,
  "loc" :  [
    - 74.016323 ,
    40.710537
  ]
}
```

- O `_id`campo contém o CEP como uma sequência.
- O `city`campo contém o nome da cidade. Uma cidade pode ter mais de um CEP associado, uma vez que diferentes seções da cidade podem ter um CEP diferente.
- O `state`campo contém a abreviação de estado de duas letras.

- O popcampo mantém a população.
DocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentação
- The `loc` field holds the location as a latitude longitude pair.

`aggregate()` Method

All of the following examples use the `aggregate()` helper in the `mongo` shell.

The `aggregate()` method uses the aggregation pipeline to processes documents into aggregated results. An aggregation pipeline consists of stages with each stage processing the documents as they pass along the pipeline. Documents pass through the stages in sequence.

The `aggregate()` method in the `mongo` shell provides a wrapper around the `aggregate` database command. See the documentation for your driver for a more idiomatic interface for data aggregation operations.

Return States with Populations above 10 Million

The following aggregation operation returns all states with total population greater than 10 million:

```
db.zipcodes.aggregate( [
   { $group: { _id: "$state", totalPop: { $sum: "$pop" } } },
   { $match: { totalPop: { $gte: 10*1000*1000 } } }
] )
```

In this example, the aggregation pipeline consists of the `$group` stage followed by the `$match` stage:

- The `$group` stage groups the documents of the `zipcode` collection by the `state` field, calculates the `totalPop` field for each state, and outputs a document for each unique state.
  The new per-state documents have two fields: the `_id` field and the `totalPop` field. The `_id` field contains the value of the `state`; i.e. the group by field. The `totalPop` field is a calculated field that contains the total population of each state. To calculate the value, `$group` uses the `$sum` operator to add the population field (`pop`) for each state.
  After the `$group` stage, the documents in the pipeline resemble the following:

```
{
  "_id" : "AK",
  "totalPop" : 550043
}
```

Documentação Documentação Documentação Documentação Documentação Documentação

- The `$match` stage filters these grouped documents to output only those documents whose `totalPop` value is greater than or equal to 10 million. The `$match` stage does not alter the matching documents but outputs the matching documents unmodified.

The equivalent SQL for this aggregation operation is:

```
SELECT state, SUM(pop) AS totalPop
FROM zipcodes
GROUP BY state
HAVING totalPop >= (10*1000*1000)
```

SEE ALSO:

`$group`, `$match`, `$sum`

Return Average City Population by State

The following aggregation operation returns the average populations for cities in each state:

```
db.zipcodes.aggregate( [
   { $group: { _id: { state: "$state", city: "$city" }, pop: { $sum: "$pop" } } },
   { $group: { _id: "$_id.state", avgCityPop: { $avg: "$pop" } } }
] )
```

In this example, the aggregation pipeline consists of the `$group` stage followed by another `$group` stage:

- The first `$group` stage groups the documents by the combination of `city` and `state`, uses the `$sum` expression to calculate the population for each combination, and outputs a document for each `city` and `state` combination. [1]
  After this stage in the pipeline, the documents resemble the following:

DocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentação

```
{
  "_id" : {
    "state" : "CO",
    "city" : "EDGEWATER"
  },
  "pop" : 13154
}
```

- A second `$group` stage groups the documents in the pipeline by the `_id.state` field (i.e. the `state` field inside the `_id` document), uses the `$avg` expression to calculate the average city population (`avgCityPop`) for each state, and outputs a document for each state.

The documents that result from this aggregation operation resembles the following:

```
{
  "_id" : "MN",
  "avgCityPop" : 5335
}
```

SEE ALSO:

`$group`, `$sum`, `$avg`

Return Largest and Smallest Cities by State

The following aggregation operation returns the smallest and largest cities by population for each state:

```
db.zipcodes.aggregate( [
   { $group:
      {
        _id: { state: "$state", city: "$city" },
        pop: { $sum: "$pop" }
      }
   },
   { $sort: { pop: 1 } },
   { $group:
      {
        _id : "$_id.state",
        biggestCity:  { $last: "$_id.city" },
        biggestPop:   { $last: "$pop" },
        smallestCity: { $first: "$_id.city" },
        smallestPop:  { $first: "$pop" }
      }
   },

  // the following $project is optional, and
  // modifies the output format.

  { $project:
    { _id: 0,
      state: "$_id",
      biggestCity:  { name: "$biggestCity",  pop: "$biggestPop" },
      smallestCity: { name: "$smallestCity", pop: "$smallestPop" }
    }
  }
] )
```

In this example, the aggregation pipeline consists of a `$group` stage, a `$sort` stage, another `$group` stage, and a `$project` stage:

- The first `$group` stage groups the documents by the combination of the `city` and `state`, calculates the `sum` of the `pop` values for each combination, and outputs a document for each `city` and `state` combination.
  At this stage in the pipeline, the documents resemble the following:

DocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentação

```
{
  "_id" : {
    "state" : "CO",
    "city" : "EDGEWATER"
  },
  "pop" : 13154
}
```

- The `$sort` stage orders the documents in the pipeline by the `pop` field value, from smallest to largest; i.e. by increasing order. This operation does not alter the documents.
- The next `$group` stage groups the now-sorted documents by the `_id.state` field (i.e. the `state` field inside the `_id` document) and outputs a document for each state.
  The stage also calculates the following four fields for each state. Using the `$last` expression, the `$group` operator creates the `biggestCity` and `biggestPop` fields that store the city with the largest population and that population. Using the `$first` expression, the `$group` operator creates the `smallestCity` and `smallestPop` fields that store the city with the smallest population and that population.
  The documents, at this stage in the pipeline, resemble the following:

```
{
  "_id" : "WA",
  "biggestCity" : "SEATTLE",
  "biggestPop" : 520096,
  "smallestCity" : "BENGE",
  "smallestPop" : 2
}
```

- The final `$project` stage renames the `_id` field to `state` and moves the `biggestCity`, `biggestPop`, `smallestCity`, and `smallestPop` into `biggestCity` and `smallestCity` embedded documents.

The output documents of this aggregation operation resemble the following:

```
{
  "state" : "RI" ,
  "biggestCity" : {
    "name" : "CRANSTON" ,
    "pop" : 176404
  },
  "smallestCity" : {
    "name" : "CLAYVILLE" ,
    "pop" : 45
  }
}
```

DocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentação

[1] Uma cidade pode ter mais de um CEP associado, uma vez que diferentes seções da cidade podem ter um CEP diferente.