Discussion 11
MongoDB & JSON
Zhe Wang
01D 10:45 6134
02D 2:15 1856
MongoDB Query

• Selection ($match), projection ($project), sorting ($sort)

• Computing/adding attributes with generalized projection ($project/$addFields), unnesting embedded arrays ($unwind), and restructuring output ($replaceRoot)

• Operators to transform/filter arrays ($map/$filter)

• Join ($lookup)

• Grouping and aggregation ($group)

Operators to aggregate (e.g., $sum) or collect into an array ($push)
Aggregation

An aggregation pipeline consists of one or more stages that process documents:

Each stage performs an operation on the input documents. For example, a stage can filter documents, group documents, and calculate values.

The documents that are output from a stage are passed to the next stage.

An aggregation pipeline can return results for groups of documents. For example, return the total, average, maximum, and minimum values.
**db.collection.aggregate**

*db.collection.aggregate*(pipeline, options). For available options, please see official documentation linked.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipeline</td>
<td>array</td>
<td>A sequence of data aggregation operations or stages. See the aggregation pipeline operators for details. The method can still accept the pipeline stages as separate arguments instead of as elements in an array; however, if you do not specify the pipeline as an array, you cannot specify the options parameter.</td>
</tr>
<tr>
<td>options</td>
<td>document</td>
<td>Optional. Additional options that <em>aggregate()</em> passes to the <em>aggregate</em> command. Available only if you specify the <em>pipeline</em> as an array.</td>
</tr>
</tbody>
</table>
Example

db.orders.aggregate([ // Stage 1: Filter pizza order documents by pizza size
  {
    $match: { size: "medium" }
  },
  // Stage 2: Group remaining documents by pizza name and calculate total quantity
  {
    $group: {
      _id: "$name",
      totalQuantity: { $sum: "$quantity" }
    }
  }
])

The $match stage:

- Filters the pizza order documents to pizzas with a size of medium.
- Passes the remaining documents to the $group stage.

The $group stage:

- Groups the remaining documents by pizza name.
- Uses $sum to calculate the total order quantity for each pizza name. The total is stored in the totalQuantity field returned by the aggregation pipeline.
Unwind: $unwind

Deconstructs an array field from the input documents to output a document for each element. Each output document is the input document with the value of the array field replaced by the element.

$unwind:

{
    path: <field path>,
    includeArrayIndex: <string>,
    preserveNullAndEmptyArrays: <boolean>
}

Example:
$unwind: {
    path: "$subcommittees"
}
Join

```javascript
db.committees.aggregate(
  [
    {
      $addFields: {
        chair_member: {
          $filter: {
            input: "$members",
            as: "member",
            cond: { $eq: ["$$member.role", "Chairman"] }
          }
        }
      }
    },
    {
      $lookup: {
        from: "people",
        localField: "chair_member.id",
        foreignField: "_id",
        as: "chair_person"
      }
    },
    {
      $project: {
        _id: false,
        name: "$displayname",
        chair: {
          $arrayElemAt: ["$chair_person.name", 0]
        }
      }
    }
  ]
)
```

- In $lookup, `localField` specifies the attribute in the current object whose value will be used for lookup.
- `from` specifies the collection in which to look for joining objects.
- `foreignField` specifies the attribute to be joined.
- $lookup creates an attribute in the current object with the name specified by `as`, and sets it value to an array holding all joining objects.
- Non-equality joins are also possible, with more complex syntax.
Replace Root: `$replaceRoot`

Replaces the input document with the specified document. The operation replaces all existing fields in the input document, including the `_id` field.

Example:

```json
$replaceRoot: {
    newRoot: {
        committee: "$displayname",
        subcommittee: "$subcommittees.displayname"
    }
}
```
RegEx: `$regex`

Provides regular expression capabilities for pattern matching strings in queries.

```json
{  
  <field>: {  
    $regex: /pattern/,  
    $options: '<options>'  
  }  
}
{  
  <field>: {  
    $regex: 'pattern',  
    $options: '<options>'  
  }  
}
{  
  <field>: {  
    $regex: /pattern/<options>  
  }  
}
```

Example:

Find legislators whose first name is Mark.

```json
{
  name: {$regex: /^Mark /}
}
```
Practice Problem

Consider a database consists of two collections

**People**: Each person element under people stores information about a legislator, including the roles he or she has served in the Congress. A role with type “rep” indicates a Representative (member of the House), while a role with type “sen” indicates a Senator (member of the Senate). A role is current if its current attribute equals 1.

**Committees**: Each committee element under committees stores information about a committee. It has a list of members, whose ids reference those of person elements in the first collection; role specifies the role of the member in the committee (e.g., chair or ranking member). Oftentimes a committee can have subcommittees. Each subcommittee element has its own list of members, which should be a subset of the committee members. A legislator can serve on multiple committees, and even multiple subcommittees under the same committee.
Q1. List all legislators born after 1972-01-01, and sort them by name in lexicographically ascending order. You can use attr: {$gte: ISODate("1972-01-01")}
Output format: [{"birthday": ...,"gender" : "...", "name" : "..."},...]

db.people.aggregate([{
  $match: {
    birthday: { $gte: ISODate("1972-01-01") },
  }},
  {$project: {
    _id: false,
    name: 1,
    birthday: 1,
    Gender: 1
  }},
  {$sort: {
    name: 1
  } }
]).toArray()
Q2. Find the number of members in each committee (not including members in the subcommittees) and sort them in descending order. Only output the top 5.

db.committees.aggregate([  
  { $project: {"_id":1, "displayname":1, "members":1}},  
  { $unwind:"$members"},  
  { $group: {    
      _id: {    
        "_id": "$_id",    
        "displayname": "$displayname",    
      },    
      count: { $sum: 1},  
    }},  
  { $project: {    
      _id: "$_id._id",    
      displayname: "$_id.displayname",    
      count: true,  
    }},  
  { $sort:{count:-1}},  
  { $limit:5},  
]).toArray()
Q3. Find the number of CURRENT (the element “current” under people.roles must be 1) Female legislators for each party.

db.people.aggregate([{
    $unwind: "$roles"},
    { $match: { "roles.current": 1, "gender": "F" }},
    { $group: {
        _id: {
            party: "$roles.party",
        },
        count: { $sum: 1 },
    }},
    { $project: {
        _id: false,
        party: "$_id.party",
        count: true,
    }}],).toArray()