Agregação com dados de preferência do usuário

Nesta página

Modelo de dados ¶

Considere um clube esportivo hipotético com um banco de dados que contenha uma `users`coleção que rastreie as datas de ingresso do usuário, as preferências esportivas e armazene esses dados em documentos semelhantes aos seguintes:

```
{
  _id  :  "jane" ,
  ingressou em  :  ISODate ( "2011-03-02" ),
  gosta de  :  [ "golf" ,  "racquetball" ]
}
{
  _id  :  "joe" ,
  ingressou em  :  ISODate ( "2012-07-02" ),
  curtidas  :  [ "tênis" ,  "golfe" ,  "natação" ]
}
```

Normalizar e classificar documentos

The following operation returns user names in upper case and in alphabetical order. The aggregation includes user names for all documents in the `users` collection. You might do this to normalize user names for processing.

```
db.users.aggregate(
  [
    { $project : { name:{$toUpper:"$_id"} , _id:0 } },
    { $sort : { name : 1 } }
  ]
)
```


All documents from the `users` collection pass through the pipeline, which consists of the following operations:

- The `$project` operator:
  - creates a new field called `name`.
  - converts the value of the `_id` to upper case, with the `$toUpper` operator. Then the `$project` creates a new field, named `name` to hold this value.
  - suppresses the `id` field. `$project` will pass the `_id` field by default, unless explicitly suppressed.
- The `$sort` operator orders the results by the `name` field.

The results of the aggregation would resemble the following:

```
{
  "name" : "JANE"
},
{
  "name" : "JILL"
},
{
  "name" : "JOE"
}
```

Return Usernames Ordered by Join Month

The following aggregation operation returns user names sorted by the month they joined. This kind of aggregation could help generate membership renewal notices.

```
db.users.aggregate(
  [
    { $project :
       {
         month_joined : { $month : "$joined" },
         name : "$_id",
         _id : 0
       }
    },
    { $sort : { month_joined : 1 } }
  ]
)
```

The pipeline passes all documents in the `users` collection through the following operations:

- The `$project` operator:
  - Creates two new fields: `month_joined` and `name`.
  - Suppresses the `id` from the results. The `aggregate()` method includes the `_id`, unless explicitly suppressed.
- The `$month` operator converts the values of the `joined` field to integer representations of the month. Then the `$project` operator assigns those values to the `month_joined` field.
- The `$sort` operator sorts the results by the `month_joined` field.

The operation returns results that resemble the following:

```
{ DocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentação
  "month_joined" : 1,
  "name" : "ruth"
},
{
  "month_joined" : 1,
  "name" : "harold"
},
{
  "month_joined" : 1,
  "name" : "kate"
}
{
  "month_joined" : 2,
  "name" : "jill"
}
```

Return Total Number of Joins per Month

The following operation shows how many people joined each month of the year. You might use this aggregated data for recruiting and marketing strategies.

```
db.users.aggregate(
  [
    { $project : { month_joined : { $month : "$joined" } } } ,
    { $group : { _id : {month_joined:"$month_joined"} , number : { $sum : 1 } } },
    { $sort : { "_id.month_joined" : 1 } }
  ]
)
```

The pipeline passes all documents in the `users` collection through the following operations:

- The `$project` operator creates a new field called `month_joined`.
- The `$month` operator converts the values of the `joined` field to integer representations of the month. Then the `$project` operator assigns the values to the `month_joined` field.
- The `$group` operator collects all documents with a given `month_joined` value and counts how many documents there are for that value. Specifically, for each unique value, `$group` creates a new “per-

_month" document with two fields:
DocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentação
    - `_id`, which contains a nested document with the `month_joined` field and its value.
    - `number`, which is a generated field. The `$sum` operator increments this field by `1` for every document containing the given `month_joined` value.
- The `$sort` operator sorts the documents created by `$group` according to the contents of the `month_joined` field.

The result of this aggregation operation would resemble the following:

```
{
  "_id" : {
    "month_joined" : 1
  },
  "number" : 3
},
{
  "_id" : {
    "month_joined" : 2
  },
  "number" : 9
},
{
  "_id" : {
    "month_joined" : 3
  },
  "number" : 5
}
```

Return the Five Most Common “Likes”

The following aggregation collects top five most “liked” activities in the data set. This type of analysis could help inform planning and future development.

```
db.users.aggregate(
  [
    { $unwind : "$likes" },
    { $group : { _id : "$likes" , number : { $sum : 1 } } },
    { $sort : { number : -1 } },
    { $limit : 5 }
  ]
)
```

The pipeline begins with all documents in the `users` collection, and passes these documents through the following operations:

- The `$unwind` operator separates each value in the `likes` array, and creates a new version of the source document for every element in the array.

  EXAMPLE:

  Given the following document from the `users` collection:

  ```
  {
    _id : "jane",
    joined : ISODate("2011-03-02"),
    likes : ["golf", "racquetball"]
  }
  ```

  The `$unwind` operator would create the following documents:

  ```
  {
    _id : "jane",
    joined : ISODate("2011-03-02"),
    likes : "golf"
  }
  {
    _id : "jane",
    joined : ISODate("2011-03-02"),
    likes : "racquetball"
  }
  ```

DocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentaçãoDocumentação

- The `$group` operator collects all documents with the same value for the `likes` field and counts each grouping. With this information, `$group` creates a new document with two fields:
  - `_id`, which contains the `likes` value.
  - `number`, which is a generated field. The `$sum` operator increments this field by `1` for every document containing the given `likes` value.
- The `$sort` operator sorts these documents by the `number` field in reverse order.
- The `$limit` operator only includes the first `5` result documents.

The results of aggregation would resemble the following:

```
{
  "_id" : "golf",
  "number" : 33
},
{
  "_id" : "racquetball",
  "number" : 31
},
{
  "_id" : "swimming",
  "number" : 24
},
{
  "_id" : "handball",
  "number" : 19
},
{
  "_id" : "tennis",
  "number" : 18
}
```