$ ordenação (agregação)

Nesta página

Definição

`$sort`

Classifica todos os documentos de entrada e os retorna ao pipeline em ordem classificada.

O `$sort`estágio tem a seguinte forma de protótipo:

```
{  $ sort :  {  < campo1 >:  < ordem de classificação  > , < campo2 >: < ordem de c
```

`$sort`pega um documento que especifica os campos a serem classificados e a respectiva ordem de classificação. pode ter um dos seguintes valores:`<sort order>`

- `1` para especificar a ordem crescente.
- `-1` para especificar a ordem decrescente.
- `{ $meta: "textScore" }`para classificar pelos `textScore` metadados computados em ordem decrescente. Consulte Classificação de metadados para um exemplo.

Exemplos

Classificação crescente / decrescente

Para que os campos sejam classificados por, defina a ordem de classificação como `1`ou `-1`especifique uma classificação crescente ou decrescente, respectivamente, como no exemplo a seguir:

```
db.users.aggregate(
   [
     { $sort : { age : -1, posts: 1 } }
   ]
)
```

This operation sorts the documents in the `users` collection, in descending order according by the `age` field and then in ascending order according to the value in the `posts` field.

When comparing values of different BSON types, MongoDB uses the following comparison order, from lowest to highest:

1. MinKey (internal type)
2. Null
3. Numbers (ints, longs, doubles, decimals)
4. Symbol, String
5. Object
6. Array
7. BinData
8. ObjectId
9. Boolean
10. Date
11. Timestamp
12. Regular Expression
13. MaxKey (internal type)

For details on the comparison/sort order for specific types, see Comparison/Sort Order.

Metadata Sort

Specify in the `{ <sort-key> }` document, a new field name for the computed metadata and specify the `$meta` expression as its value, as in the following example:

```
db.users.aggregate(
   [
     { $match: { $text: { $search: "operating" } } },
     { $sort: { score: { $meta: "textScore" }, posts: -1 } }
   ]
)
```

This operation uses the `$text` operator to match the documents, and then sorts first by the `"textScore"` metadata and then by descending order of the `posts` field. The specified metadata determines the sort order. For example, the `"textScore"` metadata sorts in descending order. See `$meta` for more information on metadata.

`$sort` Operator and Memory

`$sort` + `$limit` Memory Optimization

When a `$sort` precedes a `$limit` and there are no intervening stages that modify the number of documents, the optimizer can coalesce the `$limit` into the `$sort`. This allows the `$sort` operation to only maintain the top `n` results as it progresses, where `n` is the specified limit, and ensures that MongoDB only needs to store `n` items in memory. This optimization still applies when `allowDiskUse` is `true` and the `n` items exceed the aggregation memory limit.

Optimizations are subject to change between releases.

`$sort` and Memory Restrictions

The `$sort` stage has a limit of 100 megabytes of RAM. By default, if the stage exceeds this limit, `$sort` will produce an error. To allow for the handling of large datasets, set the `allowDiskUse` option to `true` to enable `$sort` operations to write to temporary files. See the `allowDiskUse` option in `db.collection.aggregate()` method and the `aggregate` command for details.

`$sort` Operator and Performance

`$sort` operator can take advantage of an index when placed at the **beginning** of the pipeline or placed **before** the `$project`, `$unwind`, and `$group` aggregation operators. If `$project`, `$unwind`, or `$group` occur prior to the `$sort` operation, `$sort` cannot use any indexes.

SEE ALSO:

Agregação com o conjunto de dados de CEI