How can multiple fields be used for grouping in MongoDB?

Authored by

stats writer

July 2, 2024

MongoDB is a document-oriented database that allows for flexible and efficient data grouping through the use of multiple fields. This means that data can be organized and grouped based on different criteria, such as location, date, or category, by specifying multiple fields within a single query. This enables users to perform more complex and specific data analysis, as well as retrieve data in a more structured and meaningful way. By utilizing multiple fields for grouping, MongoDB offers a versatile and powerful solution for managing and organizing data.

MongoDB: Group By Multiple Fields

You can use the following syntax to group by multiple fields and perform some aggregation in MongoDB:

db.collection.aggregate()

The following examples show how to use this syntax with a collection teams with the following documents:

db.teams.insertOne({team: "Mavs", position: "Guard", points: 31})db.teams.insertOne({team: "Mavs", position: "Guard", points: 22})db.teams.insertOne({team: "Mavs", position: "Forward", points: 19})db.teams.insertOne({team: "Rockets", position: "Guard", points: 26})db.teams.insertOne({team: "Rockets", position: "Forward", points: 33})

Example 1: Group By Multiple Fields & Aggregate

We can use the following code to group by 'team' and

'position' and count the occurrences of each grouping:

db.teams.aggregate()

This returns the following results:

{ _id: { team: 'Rockets', position: 'Forward' }, count: 1 }
{ _id: { team: 'Mavs', position: 'Guard' }, count: 2 }
{ _id: { team: 'Mavs', position: 'Forward' }, count: 1 }
{ _id: { team: 'Rockets', position: 'Guard' }, count: 1 }

We could also perform a different aggregation. For example, we could group by 'team' and position' and find the sum of 'points' by grouping:

db.teams.aggregate()

This returns the following results:

{ _id: { team: 'Rockets', position: 'Forward' }, sumPoints: 33 }
{ _id: { team: 'Mavs', position: 'Guard' }, sumPoints: 53 }
{ _id: { team: 'Mavs', position: 'Forward' }, sumPoints: 19 }
{ _id: { team: 'Rockets', position: 'Guard' }, sumPoints:

26 }

This tells us:

The sum of points scored by players on the 'Rockets' in position 'Forward' is 33.The sum of points scored by players on the 'Mavs' in position 'Guard' is 53.

And so on.

Example 2: Group By Multiple Fields & Aggregate (Then Sort)

db.teams.aggregate()

This returns the following results:

{ _id: { team: 'Mavs', position: 'Forward' }, sumPoints: 19 }
{ _id: { team: 'Rockets', position: 'Guard' }, sumPoints: 26 }
{ _id: { team: 'Rockets', position: 'Forward' }, sumPoints: 33 }
{ _id: { team: 'Mavs', position: 'Guard' }, sumPoints: 53 }

We can use -1 to instead sort the results by points in

descending order:

db.teams.aggregate()

This returns the following results:

{ _id: { team: 'Mavs', position: 'Guard' }, sumPoints: 53 }
{ _id: { team: 'Rockets', position: 'Forward' }, sumPoints: 33 }
{ _id: { team: 'Rockets', position: 'Guard' }, sumPoints: 26 }
{ _id: { team: 'Mavs', position: 'Forward' }, sumPoints: 19 }

Note: You can find the complete documentation for $group .