How to Get a Random Sample of Documents in MongoDB Using $sample

Authored by

stats writer

November 30, 2025

ARABPSYCHOLOGY.COM

RECOMMENDED CITATION

stats writer (2025). *How to Get a Random Sample of Documents in MongoDB Using $sample*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=102425

MongoDB provides the powerful $sample operator, which is specifically designed to efficiently select a representative, random sample of documents from any given collection. This operator is utilized within the aggregation pipeline and requires a single crucial configuration: the **size** parameter.

The **size** parameter dictates the exact number of documents the user wishes to retrieve randomly. The selection mechanism relies on a robust pseudo-random number generator, ensuring that the selection is statistically unbiased across the entire dataset. Furthermore, the $sample operator can be combined seamlessly with other pipeline stages, such as $match for pre-filtering, $sort for ordering the results, or $project for refining the output fields.

The Role of Sampling in Data Management

In modern data analysis and application development, dealing with massive volumes of data is commonplace. MongoDB, as a scalable NoSQL solution, frequently handles collections containing millions or even billions of documents. When performing tasks such as performance testing, building machine learning models, or conducting preliminary statistical studies, querying the entire dataset can be resource-intensive and time-consuming.

This necessity drives the requirement for effective random sampling. Random sampling allows developers and analysts to work with a smaller, statistically meaningful subset of the data. By using a truly random selection method, the properties observed in the sample are highly likely to reflect the properties of the overall population, leading to faster prototyping and development cycles without compromising the validity of initial findings.

The dedicated $sample operator within the aggregation pipeline provides a superior method compared to manual attempts at random selection, such as combining $sort with a random value and $limit. This dedicated operator is highly optimized for performance, especially in sharded cluster environments, making it the preferred tool for obtaining random subsets of data efficiently.

Implementing Random Selection with the $sample Operator

The $sample operator is an aggregation stage that must be passed into the `db.collection.aggregate()` method. Its primary function is to randomly select a specified count of input documents and pass them forward into the next stage of the pipeline, or return them as the final result if it is the last stage.

The mechanism employed by MongoDB for this selection involves a high-quality pseudo-random number generator (PRNG). This PRNG ensures that every document has an equal probability of being selected, regardless of its physical location on disk or its insertion order. This randomness is

ARABPSYCHOLOGY.COM

crucial for maintaining the integrity of the sample.

Understanding the `size` parameter is key to using this operator effectively. If the specified `size` is greater than the total number of documents available in the collection, the operator will simply return all documents, halting the process without error. If the size is zero or negative, the stage will return zero documents. Careful consideration of the required sample size relative to the overall collection size is necessary for optimizing execution time.

Core Syntax and Structure

To initiate the sampling process, one must structure an aggregation pipeline that includes the `$sample` stage. Since aggregation stages are defined as an array of operations, the basic syntax involves nesting the `$sample` operator with its required parameters inside a document within that array.

The configuration object for `$sample` must contain the key `size`. The corresponding value must be a positive integer representing the count of documents desired in the resulting sample. This structure is universally applied, whether the collection is small, large, or sharded.

The standard syntax for retrieving a specified number of documents from a collection using the **$sample** stage is illustrated below. This example selects a random sample of **4** documents from a collection named myCollection:

db.myCollection.aggregate()

By simply changing the value provided to the **size** argument, you can adjust the volume of the sample returned to suit different analytical needs, such as increasing the size for higher statistical confidence or decreasing it for faster data exploration.

Preparing the Demonstration Dataset

To clearly illustrate the functionality and non-deterministic nature of the $sample operator, we will establish a small, manageable collection called teams. This collection contains seven distinct documents, each representing a sports team and their current point totals.

The use of a small dataset allows us to visually confirm that the selection process is indeed random and that the result set changes upon repeat executions. This simulation mirrors real-world scenarios where sampling might be applied to subsets of data defined by specific criteria.

The following commands demonstrate the insertion of these seven documents into the teams collection, completing our setup phase before initiating the aggregation query.

ARABPSYCHOLOGY.COM

```
db.teams.insertOne({team: "Mavs", points: 31})
db.teams.insertOne({team: "Spurs", points: 22})
db.teams.insertOne({team: "Rockets", points: 19})
db.teams.insertOne({team: "Warriors", points: 26})
db.teams.insertOne({team: "Cavs", points: 33})
db.teams.insertOne({team: "Hornets", points: 30})
db.teams.insertOne({team: "Nets", points: 14})
```

Executing the First Random Sample Retrieval

With the dataset prepared, we can now proceed to execute the first aggregation query using the MongoDB shell. We aim to select **4** documents, which is approximately 57% of the total collection size, ensuring a high likelihood of variation between runs.

The command below illustrates the simple use of the $sample operator on our teams collection:

```
db.teams.aggregate()
```

Upon execution, this query will return four documents selected randomly by the internal pseudo-random number generator. The results of this first execution are shown below:

```
{ _id: ObjectId("6203ee711e95a9885e1e765d"),
team: 'Cavs',
points: 33 }
{ _id: ObjectId("6203ee711e95a9885e1e765b"),
team: 'Rockets',
points: 19 }
{ _id: ObjectId("6203ee711e95a9885e1e7659"),
team: 'Mavs',
points: 31 }
{ _id: ObjectId("6203ee711e95a9885e1e765f"),
team: 'Nets',
points: 14 }
```

The teams included in this specific random sample are:

Cavs
Rockets
Mavs

ARABPSYCHOLOGY.COM

Nets

Demonstrating Non-Deterministic Results

A key characteristic of effective random sampling is the inability to predict the exact output set on repeated executions. If we use the **$sample** function again, it will perform a fresh selection, meaning there is no guarantee that the same set of documents will be chosen. This non-deterministic behavior is vital for applications requiring true randomness.

To demonstrate this principle, suppose we select another random sample of **4** documents from the teams collection, using the identical query:

db.teams.aggregate()

This second execution returns a different combination of documents, showcasing the effectiveness of the underlying selection algorithm:

{ _id: ObjectId("6203ee711e95a9885e1e765b"),
team: 'Rockets',
points: 19 }
{ _id: ObjectId("6203ee711e95a9885e1e765f"),
team: 'Nets',
points: 14 }
{ _id: ObjectId("6203ee711e95a9885e1e765e"),
team: 'Hornets',
points: 30 }
{ _id: ObjectId("6203ee711e95a9885e1e765c"),
team: 'Warriors',
points: 26 }

The four teams included in this new random sample are:

Rockets
Nets
Hornets
Warriors

Notice that this second sample does not perfectly match the random sample from the previous example. Specifically, the Hornets and Warriors were selected this time, replacing the Cavs and Mavs, confirming that the selection process is genuinely random and unique for each query

ARABPSYCHOLOGY.COM

execution.

Combining $sample with Filtering and Projection

A powerful aspect of using the $sample operator is its ability to integrate into complex aggregation pipeline workflows. By positioning the `$sample` stage strategically, users can achieve filtered or summarized random data efficiently.

If we want to sample only documents that meet a certain condition--for example, teams with points greater than 25--we would place a $match stage immediately before the `$sample` stage. This ensures that the random selection only draws from the subset of documents that satisfy the filter criteria, guaranteeing that all sampled documents are relevant.

Conversely, if performance is paramount and the collection is enormous, one might place $sample first to reduce the document count significantly, followed by $match to filter the small sampled subset. While faster, this approach carries a small risk that the final sample might not fully represent the filtered criteria, especially if the sample size is minimal. The best practice is usually to filter first, then sample.

Performance and Optimization Considerations

For very large collections, the efficiency of the `$sample` stage is maximized when the requested size is a small fraction of the total collection. MongoDB often employs sophisticated techniques, similar to reservoir sampling, to fulfill the request without having to read the entire collection into memory, thereby minimizing I/O operations.

However, as the requested `size` increases relative to the total number of documents, the cost of the `$sample` operation grows. If the requested size approaches the collection size, the optimization benefits diminish, and the operation essentially reverts to a full collection scan to ensure all available documents are included in the pool for random selection.

For development and analysis environments, it is good practice to use the `.explain("executionStats")` method to analyze the performance of the aggregation query, ensuring that the `$sample` stage is operating optimally, particularly when dealing with sharded clusters where data distribution affects randomness and read efficiency.

Note: You can find the complete documentation for the **$sample** function on the official MongoDB documentation site.

Conclusion

The $sample operator is an indispensable tool for data manipulation in MongoDB, providing a

ARABPSYCHOLOGY.COM

simple yet powerful means to obtain random, unbiased subsets of documents. By understanding its integration within the aggregation pipeline and the role of the **size** parameter, developers can effectively manage and analyze large datasets without prohibitive computational costs.

The ability to generate samples quickly and reliably using a pseudo-random number generator ensures that analytical results derived from the sample are statistically sound, accelerating research and development cycles significantly.

The following tutorials explain how to perform other common operations in MongoDB:

MongoDB: How to Query for "NOT NULL" in Specific Field

ARABPSYCHOLOGY.COM