How to Round Decimal Places in MongoDB Using the $round Operator

Authored by

stats writer

November 30, 2025

In modern data management, maintaining precision while ensuring data readability is a frequent requirement. When dealing with complex numerical data, especially financial metrics, scores, or scientific measurements, it often becomes necessary to standardize values by rounding them to a specific number of decimal places.

Fortunately, MongoDB, a leading NoSQL database, offers robust tools within its Aggregation Pipeline to handle such numerical transformations efficiently. The primary mechanism for this is the powerful **$round operator**, designed specifically for controlling how numeric values are rounded.

This operator is essential for preparing data for reporting, display, or subsequent calculations where strict precision rules must be followed. By understanding how to properly implement **$round** within an aggregation stage, developers can ensure their results are both accurate and formatted correctly for end-users or downstream systems.

Understanding the $round operator Syntax

The **$round operator** is a crucial component of the Aggregation Pipeline, allowing for mathematical manipulation of fields. It is typically used within the **$project** stage, which allows you to reshape documents and create new calculated fields based on existing data.

The **$round** operator requires an array of arguments. The first argument specifies the numeric expression or field containing the value to be rounded. The second, optional argument specifies the integer number of decimal places to round to. If the second argument is omitted, **$round** defaults to rounding to the nearest whole number (zero decimal places).

This functionality is foundational for tasks requiring controlled numerical output in MongoDB. The following basic syntax demonstrates its structure, where `$value` represents the field name and `1` represents the target number of decimal places:

db.myCollection.aggregate(}}}])

As shown in the syntax above, we are utilizing the **$project** stage to define a new field called `rounded_value`, which is populated by applying the **$round operator** to the source field `$value`. This particular example rounds the values in the field "value" to one decimal place.

Setting Up the Example Data Collection

To illustrate the practical application of the **$round** operator, we will use a sample collection named `teams`. This collection simulates score data for various sports teams, where the `points` field contains numerical values with varying degrees of precision.

The initial setup involves inserting several documents into the `teams` collection. These documents include the team name and their average or accumulated points, which we will subsequently manipulate using the Aggregation Pipeline.

The following insertion commands define our dataset:

db.teams.insertOne({team: "Mavs", points: 31.345})
db.teams.insertOne({team: "Spurs", points: 22.88})
db.teams.insertOne({team: "Rockets", points: 19.91})
db.teams.insertOne({team: "Warriors", points: 26.5})
db.teams.insertOne({team: "Cavs", points: 33})

Notice the different levels of precision in the `points` field--from three decimal places (31.345) to integers (33). Our goal is to use the **$round** operator to standardize the presentation of these scores.

Case Study 1: Rounding Values to One Decimal Place

A very common requirement is ensuring data is presented consistently, often limited to a specific precision, such as one decimal place for financial or statistical reports. By specifying `1` as the second argument to the **$round operator**, we instruct MongoDB to adhere to this rounding standard.

We will execute the following Aggregation Pipeline command. This pipeline uses the **$project** stage to create a new field, `rounded_points`, based on the rounded value of the original `$points` field:

db.teams.aggregate(}}}])

When this aggregation is executed, we can observe how the rounding rules are applied. Standard half-up rounding is used. For example, 31.345 rounds down to 31.3, while 22.88 rounds up to 22.9.

Here is the resulting output from the command execution:

{ _id: ObjectId("6203d3b11e95a9885e1e763b"),
rounded_points: 31.3 }
{ _id: ObjectId("6203d3b11e95a9885e1e763c"),
rounded_points: 22.9 }
{ _id: ObjectId("6203d3b11e95a9885e1e763d"),
rounded_points: 19.9 }
{ _id: ObjectId("6203d3b11e95a9885e1e763e"),

rounded_points: 26.5 }
{ _id: ObjectId("6203d3b11e95a9885e1e763f"),
rounded_points: 33 }

A careful review of the output confirms that every value in the `rounded_points` field is now consistently formatted to include at most one digit after the decimal point, demonstrating effective data standardization using the **$round** operator.

Case Study 2: Default Rounding to the Nearest Integer

If the goal is to obtain whole numbers, or integers, from floating-point values--effectively rounding to zero decimal places--we have two primary options when using the **$round** operator. We can explicitly set the second argument to `0`, or we can omit the second argument entirely, allowing MongoDB to apply the default behavior.

The omission of the decimal places argument is particularly useful for streamlined code when integer conversion is the primary objective. This simplifies the syntax while achieving the same mathematical result as passing `0`.

For example, suppose we use the **$round operator** with the following syntax, deliberately leaving out the decimal precision specification:

db.teams.aggregate(}}}])

The Aggregation Pipeline processes the `points` field, determining the nearest integer for each value. Values equal to or greater than X.5 are rounded up, while values less than X.5 are rounded down.

Here is the output, showcasing the result of the default integer rounding:

{ _id: ObjectId("6203d3b11e95a9885e1e763b"),
rounded_points: 31 }
{ _id: ObjectId("6203d3b11e95a9885e1e763c"),
rounded_points: 23 }
{ _id: ObjectId("6203d3b11e95a9885e1e763d"),
rounded_points: 20 }
{ _id: ObjectId("6203d3b11e95a9885e1e763e"),
rounded_points: 26 }
{ _id: ObjectId("6203d3b11e95a9885e1e763f"),
rounded_points: 33 }

Notice how values like 31.345 are rounded down to 31, 22.88 is rounded up to 23, and 19.91 is rounded up to 20. This confirms that the default behavior of the **$round** operator is consistent with standard mathematical rounding to the nearest integer.

Advanced Projection: Including Source Data and Other Fields

When using the Aggregation Pipeline, particularly the **$project** stage, it is often necessary to display both the original, unrounded value and the newly calculated rounded value side-by-side, along with other descriptive fields like the team name.

The **$project** stage controls which fields are passed to the next stage or returned to the user. We use a value of **1** (to display) or **0** (to suppress) to explicitly manage the output structure. By default, the `_id` field is included unless explicitly suppressed by setting `_id: 0`.

For example, the following code shows how to round the points values to two decimal places while simultaneously displaying the original `points` field and the descriptive `team` field. We also suppress the default `_id` field for cleaner output:

```
db.teams.aggregate(
}
}
}
]
)
```

This detailed projection demonstrates how to construct complex output documents that combine original data with transformation results, which is essential for audit trails or comparative analysis.

Here is the resulting output, clearly showing the team, the original points, and the new points rounded to two decimal places:

```
{ team: 'Mavs', points: 31.345, rounded_points: 31.34 }
{ team: 'Spurs', points: 22.88, rounded_points: 22.88 }
{ team: 'Rockets', points: 19.91, rounded_points: 19.91 }
{ team: 'Warriors', points: 26.5, rounded_points: 26.5 }
{ team: 'Cavs', points: 33, rounded_points: 33 }
```

Considering Negative Precision and Data Types

The **$round** operator offers further flexibility by allowing negative values for the decimal precision

argument. Using a negative integer instructs MongoDB to round to the nearest power of 10. For instance, passing `-1` rounds to the nearest ten, and passing `-2` rounds to the nearest hundred. This is extremely useful for generating estimated or summarized large numbers.

It is also critical to understand how **$round** handles various BSON data types. While **$round** is primarily designed for numeric types (Double, Decimal128, Int32, Int64), if the input value is null, missing, or non-numeric, the operator will return `null`. If the input is a 32-bit integer, and no rounding is necessary (i.e., rounding to 0 or negative precision), the output remains an integer, maintaining performance benefits where possible.

However, if the rounding operation results in a value that must be represented as a floating-point number, the output type may change. For robust development, it is often best practice to use the Decimal128 type for financial or sensitive data where exact decimal arithmetic is required, though **$round** works effectively across most numeric types.

Related Operators: `$ceil` and `$floor`

While **$round** performs standard mathematical rounding (rounding half up), MongoDB provides other specialized operators for different rounding needs within the Aggregation Pipeline. These include **$ceil** and **$floor**.

The **$ceil** operator (ceiling) returns the smallest integer greater than or equal to the specified numeric expression. This is useful when you always need to bump a value up to the next whole unit, regardless of the fractional part (e.g., calculating bandwidth usage or allocating resources).

Conversely, the **$floor** operator returns the largest integer less than or equal to the specified numeric expression. This is equivalent to truncation toward negative infinity and is often used to discard fractional parts entirely (e.g., age calculation or integer pricing). Unlike **$round**, neither **$ceil** nor **$floor** supports specifying a number of decimal places; they always return an integer result.

Summary and Official Documentation

The **$round operator** is the definitive solution for precision control in MongoDB. By integrating it into the **$project** stage of the Aggregation Pipeline, developers can effectively standardize numeric data, ensuring consistency and accuracy in reporting and application display.

Note: You can find the complete documentation for the **$round** function on the official MongoDB website, which provides exhaustive details on argument handling, data type implications, and behavior edge cases.

The ability to easily manipulate numerical precision within the database layer significantly reduces the complexity of application logic and improves performance by leveraging MongoDB's optimized aggregation framework.

Further MongoDB Operations and Tutorials

Mastering data transformation in MongoDB extends beyond simple rounding. The following resources explain how to perform other common operations and advanced transformations in MongoDB:

How to use date operators for time series analysis.

Techniques for string manipulation and text processing.

Implementing conditional logic using the **$cond** operator.