Operatory wyrażeń (logika i formatowanie wewnątrz etapów)

Poziom: podstawy **Dotyczy: MongoDB 5.0–8.0** **Czas: ~20 min**

Operatory, którymi wewnątrz etapów potoku agregacji liczymy wartości — warunki, działania arytmetyczne, formatowanie tekstu i dat — zamiast tylko filtrować dokumenty.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
B2B	business to business: sprzedaż i systemy skierowane do innych firm, nie do klientów indywidualnych
CSV	w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami
UTC	Coordinated Universal Time: uniwersalny czas bez strefy, punkt odniesienia dla wszystkich stref czasowych na świecie

Fiszka

Wszystkie poniższe operatory wykonują się w powłoce `mongosh`, w ramach etapu potoku agregacji.

TYLKO ODCZYT

Warunek if/then/else: $cond

```
db.zamowienia.aggregate([
  { $addFields: {
      kategoriaKwoty: {
        $cond: { if: { $gte: ["$kwota", 500] }, then: "duze", else: "male" }
```

```
      }
  } }
])
```

Zwracamy jedną z dwóch wartości w zależności od spełnienia warunku — odpowiednik `if`/`else`. Powyżej każdemu zamówieniu dopisywane jest pole `kategoriaKwoty`, równe `"duze"`, gdy kwota wynosi co najmniej 500, w przeciwnym razie `"male"`.

Skrócona składnia tablicowa, gdzie kolejność argumentów to warunek, wartość przy prawdzie i wartość przy fałszu:

```
{ $cond: [ { $gte: ["$kwota", 500] }, "duze", "male" ] }
```

Przykład:

```
[
  { _id: ObjectId('...'), kwota: 620, kategoriaKwoty: 'duze' },
  { _id: ObjectId('...'), kwota: 340, kategoriaKwoty: 'male' }
]
```

TYLKO ODCZYT

Wielokrotny wybór: $switch

```
db.zamowienia.aggregate([
  { $addFields: {
      prog: {
        $switch: {
          branches: [
            { case: { $gte: ["$kwota", 1000] }, then: "premium" },
            { case: { $gte: ["$kwota", 500] }, then: "standard" }
          ],
          default: "podstawowy"
        }
      }
  } }
])
```

Sprawdzamy kolejno warunki z listy `branches` i zwracamy wynik pierwszego pasującego. Jeśli żaden warunek nie pasuje, zwracamy wartość z `default`. To czytelniejsza alternatywa dla

kilku zagnieżdżonych `$cond` naraz. Warunki są sprawdzane od góry listy w dół, więc ich kolejność ma znaczenie — dokument o kwocie 1200 pasujący jednocześnie do obu warunków dostanie wynik pierwszego z nich, `"premium"`.

TYLKO ODCZYT

Wartość domyślna dla braków: $ifNull

```
db.zamowienia.aggregate([
  { $addFields: { rabat: { $ifNull: ["$rabat", 0] } } }
])
```

Zwracamy pierwszą wartość, jeśli nie jest ona `null` ani nieobecna w dokumencie; w przeciwnym razie zwracamy wartość zastępczą podaną jako drugi argument. Tutaj brakujące pole `rabat` zostaje uzupełnione zerem, co zabezpiecza dalsze obliczenia arytmetyczne przed błędami wynikającymi z wartości `null`.

TYLKO ODCZYT

Akumulatory grupujące: $sum, $avg, $min, $max

```
{ $group: {
    _id: "$klient",
    liczba: { $sum: 1 },
    sredniaKwota: { $avg: "$kwota" },
    minimalna: { $min: "$kwota" },
    maksymalna: { $max: "$kwota" }
} }
```

Cztery podstawowe akumulatory liczone w obrębie każdej grupy etapu `$group`, opisanego na stronie Aggregation pipeline. `$sum: 1` zlicza liczbę dokumentów w grupie, `$sum` z nazwą pola sumuje jego wartości, a `$avg`, `$min`, `$max` liczą odpowiednio średnią i wartości skrajne.

TYLKO ODCZYT

Pierwszy i ostatni dokument grupy: $first i $last

```
{ $group: { _id: "$klient", pierwszeZamowienie: { $first: "$data" }, ostatnieZamowienie: { $last: "$data" } } }
```

Zwracają wartość pola z pierwszego albo ostatniego dokumentu napotkanego w obrębie grupy — mają sens tylko wtedy, gdy dokumenty trafiają do `$group` w ustalonej kolejności, czyli po wcześniejszym `$sort` w tym samym potoku.

TYLKO ODCZYT

Zbieranie wartości do tablicy: $push i $addToSet

```
{ $group: { _id: "$klient", produkty: { $push: "$produkt" } } }
```

Zbieramy wartości pola ze wszystkich dokumentów grupy do jednej tablicy, z zachowaniem duplikatów i kolejności napotkania.

```
{ $group: { _id: "$klient", produktyUnikalne: { $addToSet: "$produkt" } } }
```

Działa identycznie jak `$push`, ale pomija duplikaty — wynikowa tablica zawiera każdą wartość tylko raz, w kolejności nieokreślonej.

TYLKO ODCZYT

Sklejanie tekstu: $concat

```
{ $concat: ["$imie", " ", "$nazwisko"] }
```

Sklejamy kilka fragmentów tekstu w jeden ciąg. **Uważaj:** jeśli którykolwiek z fragmentów ma wartość `null`, cały wynik `$concat` również będzie `null` — warto opakować niepewne pole w `$ifNull` przed sklejeniem.

TYLKO ODCZYT

Wycinek tekstu: $substr

```
{ $substr: ["$kod", 0, 3] }
```

Zwracamy fragment tekstu — pierwszy argument to pole źródłowe, drugi pozycja startowa liczona od zera, trzeci długość wycinka.

TYLKO ODCZYT

Zmiana wielkości liter: $toUpper

```
{ $toUpper: "$status" }
```

Zamieniamy wartość pola tekstowego na same wielkie litery — odpowiednik jest też `$toLower`, działający analogicznie w drugą stronę.

TYLKO ODCZYT

Formatowanie daty jako tekstu: $dateToString

```
db.zamowienia.aggregate([
  { $addFields: {
      dataTekst: {
        $dateToString: { format: "%Y-%m-%d", date: "$data", timezone: "Europe/Warsaw" }
      }
  } }
])
```

Zamieniamy pole typu data na sformatowany tekst według wzorca podanego w `format`. Opcjonalny `timezone` przelicza datę na wskazaną strefę czasową przed sformatowaniem — bez niego daty są traktowane jako UTC.

Przykład:

```
[ { _id: ObjectId('...'), data: ISODate('2026-09-15T22:30:00Z'), dataTekst: '2026-09-16'
} ]
```

UTC = Coordinated Universal Time: uniwersalny czas bez strefy, punkt odniesienia dla wszystkich stref czasowych na świecie

TYLKO ODCZYT

Różnica między dwiema datami: $dateDiff

```
{ $dateDiff: { startDate: "$data", endDate: "$$NOW", unit: "day" } }
```

Liczymy odstęp między dwiema datami w zadanej jednostce (`day`, `hour`, `month` i inne). Zmienna `$$NOW` oznacza bieżący czas serwera w chwili wykonania potoku — podwójny prefiks

`$$` odróżnia zmienną systemową albo lokalną od zwykłego odwołania do pola dokumentu.

□□ TYLKO ODCZYT

Konwersja typów: $convert i $toInt

```
{ $toInt: "$kwotaTekst" }
```

Zamieniamy tekst na liczbę całkowitą — niezbędne przy porządkowaniu danych zaimportowanych jako tekst, na przykład z pliku CSV przez `mongoimport`, opisany na stronie Backup i odtwarzanie: mongodump i mongorestore.

Bardziej elastyczny jest `$convert`, pozwalający wskazać typ docelowy i wartość zastępczą na wypadek błędu konwersji:

```
{ $convert: { input: "$kwotaTekst", to: "double", onError: 0, onNull: 0 } }
```

Bez `onError` błąd konwersji (na przykład tekst niebędący liczbą) przerywa cały potok wyjątkiem — `onError` zamienia go na spokojnie zwróconą wartość zastępczą.

□□ **CSV** = w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami

□□ TYLKO ODCZYT

Dostęp do elementu tablicy po indeksie: $arrayElemAt

```
{ $arrayElemAt: ["$produkty", 0] }
```

Zwracamy element tablicy o wskazanym indeksie, liczonym od zera. Indeks ujemny liczy od końca tablicy — `-1` zwraca ostatni element.

□□ TYLKO ODCZYT

Liczba elementów tablicy: $size

```
{ $size: "$produkty" }
```

Zwracamy liczbę elementów w tablicy — bywa przydatne po etapie `$lookup`, żeby policzyć liczbę dopasowań znalezionych w innej kolekcji bez rozwijania ich przez `$unwind`.

TYLKO ODCZYT

Filtrowanie elementów tablicy: $filter

```
{ $filter: { input: "$produkty", as: "p", cond: { $gt: ["$$p.cena", 100] } } }
```

Zwracamy podzbiór tablicy spełniający podany warunek. Zmienna pętli `$$p` (nazwa dowolna, zadeklarowana w `as`) ma podwójny prefiks `$$`, bo odnosi się do zmiennej lokalnej wprowadzonej przez sam operator, a nie do pola całego dokumentu.

TYLKO ODCZYT

Przekształcanie elementów tablicy: $map

```
{ $map: { input: "$produkty", as: "p", in: "$$p.nazwa" } }
```

Przekształcamy każdy element tablicy według wyrażenia podanego w `in` — tutaj wyciągamy samą nazwę z każdego obiektu produktu, zamieniając tablicę obiektów na tablicę samych nazw.

Z życia wzięte

Z ŻYCIA WZIĘTE

Marek prosi o raport, w którym każdy produkt w katalogu B2B ma pokazaną cenę z rabatem naliczonym warunkowo, zależnie od kategorii i ilości na stanie. Ola buduje to jednym etapem $project z zagnieżdżonym $cond zamiast liczyć rabaty po stronie aplikacji dla każdego z kilkudziesięciu tysięcy produktów osobno. Zapytanie, które w pierwszej wersji trwało kilkanaście sekund, po przeniesieniu logiki do operatora wewnątrz potoku wraca niemal natychmiast.

Po co to istnieje

Pojęcie: operator wyrażenia. Operatory wyrażeń to konstrukcje używane wewnątrz etapów potoku agregacji, opisanego na stronie Aggregation pipeline — najczęściej `$project`, `$addFields` i `$group` — do obliczania nowych wartości: warunków, działań arytmetycznych,

> formatowania tekstu i dat. Różnią się od operatorów zapytań używanych w `find()` i `$match`, które jedynie filtrują dokumenty, nie tworząc niczego nowego. Operatory wyrażeń zawsze coś obliczają i zwracają.

Kluczowa różnica składni: operatory porównań w wyrażeniach przyjmują argumenty w postaci tablicy, na przykład `{ $gt: ["$kwota", 100] }`. To nie to samo co forma zapytania `{ kwota: { $gt: 100 } }` używana w `$match` i `find()`, opisana na stronie Operacje na dokumentach. Mylenie tych dwóch form jest najczęstszym błędem, jaki popełnia się przy pierwszym kontakcie z potokiem agregacji.

Przykład złożony — operatory w akcji

```
db.zamowienia.aggregate([
  { $addFields: {
      brutto: { $round: [ { $multiply: ["$kwota", 1.23] }, 2 ] },
      miesiac: { $dateToString: { format: "%Y-%m", date: "$data" } },
      etykieta: { $cond: [ { $gte: ["$kwota", 500] }, "VIP", "zwykly" ] },
      rabat: { $ifNull: ["$rabat", 0] }
  } }
])
```

Jeden etap `$addFields` liczy tu kwotę brutto zaokrągloną do dwóch miejsc po przecinku, wyciąga miesiąc zamówienia jako tekst, nadaje etykietę na podstawie progu kwoty i uzupełnia brakujący rabat zerem. Pokazuje to, jak operatory wyrażeń składa się ze sobą, żeby w jednym przebiegu przygotować dane gotowe do dalszego raportu.

Częste błędy

Użycie formy zapytania zamiast formy wyrażenia. `{ kwota: { $gt: 100 } }` wewnątrz `$addFields` czy `$project` nie zadziała tak jak w `$match` — trzeba użyć `{ $gt: ["$kwota", 100] }`, z argumentami w tablicy.

Pojedynczy `$` zamiast `$$` przy zmiennej lokalnej. Wewnątrz `$filter` czy `$map` odwołanie do zmiennej pętli z pojedynczym `$` (`$p.cena` zamiast `$$p.cena`) MongoDB potraktuje jako próbę odwołania do pola dokumentu o nazwie `p`, którego nie ma, i zwróci `null` zamiast oczekiwanej wartości.

Konwersja bez obsługi błędu. Zwykłe `$toInt` na polu, które czasem zawiera tekst nienadający się do konwersji, przerywa cały potok wyjątkiem — w danych o niepewnej jakości bezpieczniejszy jest `$convert` z ustawionym `onError`.

Powiązane: Aggregation pipeline · Operacje na dokumentach

Źródła: notatki własne administratora MongoDB · dokumentacja MongoDB, rozdział „Aggregation Pipeline Operators"

Revision #42
Created 2026-06-09 09:41:32 UTC by Tomasz Konieczny
Updated 2026-10-26 14:10:01 UTC by Tomasz Konieczny