Operacje na dokumentach

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

Wstawianie, wyszukiwanie, aktualizowanie i usuwanie dokumentów — cztery operacje, na których stoi cała praca z danymi w MongoDB, wykonywane wprost z powłoki `mongosh`.

Skróty użyte na tej stronie (2)

Skrót	Co znaczy
B2B	business to business: sprzedaż i systemy skierowane do innych firm, nie do klientów indywidualnych
SQL	Structured Query Language: język zapytań do relacyjnych baz danych

Fiszka

Komendy wykonujemy w powłoce `mongosh`, opisanej na stronie mongosh: powłoka administracyjna, po wcześniejszym przełączeniu się na właściwą bazę poleceniem `use`.

TYLKO ODCZYT

Wstawienie pojedynczego dokumentu: insertOne

```
db.uzytkownicy.insertOne({ imie: "Anna", wiek: 30, miasto: "Warszawa" })
```

Wstawiamy jeden dokument do kolekcji. Jeśli kolekcja jeszcze nie istnieje, MongoDB tworzy ją automatycznie przy pierwszym zapisie.

Przykład:

```
{
  acknowledged: true,
  insertedId: ObjectId('66f1b2c3d4e5f6a7b8c9d0e1')
}
```

TYLKO ODCZYT

Wstawienie wielu dokumentów naraz: insertMany

```
db.uzytkownicy.insertMany([
  { imie: "Jan", wiek: 41 },
  { imie: "Ola", wiek: 22 }
])
```

Wstawiamy tablicę dokumentów jednym poleceniem — szybsze niż wywoływanie `insertOne` w pętli, bo wymaga jednej podróży do serwera zamiast wielu.

Przykład:

```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('66f1b30ad4e5f6a7b8c9d0e2'),
    '1': ObjectId('66f1b30ad4e5f6a7b8c9d0e3')
  }
}
```

TYLKO ODCZYT

Wyszukanie wszystkich dokumentów w kolekcji: find

```
db.uzytkownicy.find()
```

Zwracamy wszystkie dokumenty kolekcji bez żadnego filtra. Wynik w powłoce jest domyślnie sformatowany czytelnie, po jednym dokumencie na wpis.

Przykład:

```
[
  { _id: ObjectId('66f1b2c3d4e5f6a7b8c9d0e1'), imie: 'Anna', wiek: 30, miasto:
'Warszawa' },
  { _id: ObjectId('66f1b30ad4e5f6a7b8c9d0e2'), imie: 'Jan', wiek: 41 },
  { _id: ObjectId('66f1b30ad4e5f6a7b8c9d0e3'), imie: 'Ola', wiek: 22 }
]
```

TYLKO ODCZYT

Wyszukanie z filtrem: operator $eq

```
db.uzytkownicy.find({ imie: "Anna" })
```

Dopasowanie wartości pola wprost, bez operatora, jest równoważne użyciu operatora `$eq` — obie formy zwracają ten sam wynik, ale forma bez operatora jest krótsza i częściej spotykana w praktyce.

TYLKO ODCZYT

Wyszukanie z operatorem porównania: $gt

```
db.uzytkownicy.find({ wiek: { $gt: 25 } })
```

Zwracamy dokumenty, w których wartość pola `wiek` jest większa niż podana liczba. Operator zapisujemy jako pole wewnątrz obiektu przypisanego do filtrowanego klucza — to jest forma zapytania, różna składniowo od formy używanej wewnątrz wyrażeń potoku agregacji, opisanej na stronie Operatory wyrażeń.

Przykład:

```
[
  { _id: ObjectId('66f1b2c3d4e5f6a7b8c9d0e1'), imie: 'Anna', wiek: 30, miasto:
'Warszawa' },
  { _id: ObjectId('66f1b30ad4e5f6a7b8c9d0e2'), imie: 'Jan', wiek: 41 }
]
```

TYLKO ODCZYT

Wyszukanie z listą dopuszczalnych wartości: $in

```
db.uzytkownicy.find({ imie: { $in: ["Anna", "Jan"] } })
```

Zwracamy dokumenty, w których wartość pola pasuje do dowolnej z wartości podanych w tablicy — odpowiednik operatora `IN` z SQL.

SQL = Structured Query Language: język zapytań do relacyjnych baz danych

TYLKO ODCZYT

Wyszukanie dopasowaniem tekstowym: $regex

```
db.uzytkownicy.find({ imie: { $regex: "^An", $options: "i" } })
```

Dopasowujemy wartość pola tekstowego do wyrażenia regularnego. Opcja `i` w `$options` sprawia, że dopasowanie ignoruje wielkość liter.

TYLKO ODCZYT

Wyszukanie po obecności pola: $exists

```
db.uzytkownicy.find({ miasto: { $exists: true } })
```

Zwracamy dokumenty, w których wskazane pole w ogóle występuje, niezależnie od jego wartości — przydatne w kolekcjach bez wymuszonego schematu, gdzie część dokumentów może w ogóle nie mieć danego pola, zamiast mieć je ustawione na `null`.

TYLKO ODCZYT

Złożenie warunków spójnikiem logicznym: $and i $or

```
db.uzytkownicy.find({ $and: [ { wiek: { $gte: 18 } }, { miasto: "Warszawa" } ] })
```

Łączymy kilka warunków spójnikiem logicznym `AND` — w praktyce rzadko potrzebny jawnie, bo kilka pól podanych obok siebie w jednym obiekcie filtra jest już domyślnie łączonych przez `AND`. Jawny `$and` jest konieczny, gdy ten sam klucz musi wystąpić w filtrze więcej niż raz.

```
db.uzytkownicy.find({ $or: [ { wiek: { $lt: 18 } }, { miasto: "Kraków" } ] })
```

Odpowiednik spójnika `OR` — dokument spełnia warunek, jeśli pasuje do choćby jednego z podanych.

TYLKO ODCZYT

Ograniczenie zwracanych pól: projekcja

```
db.uzytkownicy.find({ miasto: "Warszawa" }, { imie: 1, wiek: 1, _id: 0 })
```

Drugi argument `find()` to projekcja — decyduje, które pola trafią do wyniku. Wartość `1` oznacza pokazanie pola, `0` jego ukrycie; pole `_id` jest jedynym, które można wyłączyć razem z jawnie włączonymi innymi polami.

Przykład:

```
[ { imie: 'Anna', wiek: 30 } ]
```

TYLKO ODCZYT

Aktualizacja pierwszego pasującego dokumentu: updateOne

```
db.uzytkownicy.updateOne(
  { imie: "Anna" },
  { $set: { wiek: 31 } }
)
```

Aktualizujemy pierwszy dokument pasujący do filtra. Operator `$set` ustawia wartość wskazanego pola, tworząc je, jeśli wcześniej nie istniało.

Przykład:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1
}
```

TYLKO ODCZYT

Zwiększenie wartości liczbowej: $inc

```
db.uzytkownicy.updateOne({ imie: "Anna" }, { $inc: { wiek: 1 } })
```

Zwiększamy (albo zmniejszamy, przy wartości ujemnej) liczbową wartość pola o podaną wielkość — bez konieczności najpierw odczytywać bieżącej wartości po stronie aplikacji.

🔒 TYLKO ODCZYT

Dodanie elementu do tablicy: $push

```
db.uzytkownicy.updateOne({ imie: "Anna" }, { $push: { role: "operator" } })
```

Dopisujemy nową wartość na koniec tablicy przechowywanej w polu dokumentu. Jeśli pole jeszcze nie istnieje, zostaje utworzone jako nowa tablica z jednym elementem.

🔒 TYLKO ODCZYT

Usunięcie pola z dokumentu: $unset

```
db.uzytkownicy.updateOne({ imie: "Anna" }, { $unset: { miasto: "" } })
```

Usuwamy wskazane pole z dokumentu całkowicie — wartość podana po prawej stronie nie ma znaczenia, liczy się tylko sama nazwa pola.

🔒 TYLKO ODCZYT

Aktualizacja wielu dokumentów naraz: updateMany

```
db.uzytkownicy.updateMany(
  { wiek: { $lt: 18 } },
  { $set: { pelnoletni: false } }
)
```

Aktualizujemy wszystkie dokumenty pasujące do filtra, nie tylko pierwszy — składnia operatorów aktualizacji jest identyczna jak w `updateOne`.

🔒 TYLKO ODCZYT

Aktualizacja z utworzeniem dokumentu, gdy brak dopasowania: upsert

```
db.uzytkownicy.updateOne(
  { imie: "Marek" },
  { $set: { wiek: 27 } },
```

```
  { upsert: true }
)
```

Trzeci argument `{ upsert: true }` sprawia, że jeśli żaden dokument nie pasuje do filtra, MongoDB tworzy nowy dokument łączący pola z filtra i z operatora `$set`, zamiast nic nie robić.

□□ TYLKO ODCZYT

Zastąpienie całego dokumentu: replaceOne

```
db.uzytkownicy.replaceOne(
  { imie: "Jan" },
  { imie: "Jan", wiek: 42, miasto: "Poznań" }
)
```

Zastępujemy cały dokument nowym, podanym wprost — w odróżnieniu od `updateOne` z `$set`, pola nieobecne w nowym dokumencie znikają, bo cały poprzedni dokument zostaje nadpisany, nie scalony.

□□ TYLKO ODCZYT

Usunięcie pierwszego pasującego dokumentu: deleteOne

```
db.uzytkownicy.deleteOne({ imie: "Ola" })
```

Usuwamy pierwszy dokument pasujący do filtra.

Przykład:

```
{ acknowledged: true, deletedCount: 1 }
```

□□ TYLKO ODCZYT

Usunięcie wszystkich pasujących dokumentów: deleteMany

```
db.uzytkownicy.deleteMany({ wiek: { $lt: 18 } })
```

Usuwamy wszystkie dokumenty pasujące do filtra. Pusty filtr `{}` usunąłby całą zawartość kolekcji naraz, dlatego warto zawsze świadomie sprawdzić filtr, zanim wykonamy tę operację na danych produkcyjnych.

TYLKO ODCZYT

Sortowanie, stronicowanie i ograniczenie wyniku: sort, skip, limit

```
db.uzytkownicy.find().sort({ wiek: -1 }).skip(10).limit(5)
```

Sortujemy wynik malejąco po wieku (`-1`, rosnąco byłoby `1`), pomijamy pierwsze dziesięć dopasowań i zwracamy kolejnych pięć — typowy wzorzec stronicowania wyników w interfejsie użytkownika.

TYLKO ODCZYT

Policzenie dokumentów pasujących do filtra: countDocuments

```
db.uzytkownicy.countDocuments({ miasto: "Warszawa" })
```

Zwracamy liczbę dokumentów pasujących do filtra, bez pobierania samych dokumentów — szybsze niż pobranie wyniku i policzenie go po stronie aplikacji.

Przykład:

```
3
```

TYLKO ODCZYT

Pobranie unikalnych wartości pola: distinct

```
db.uzytkownicy.distinct("miasto")
```

Zwracamy listę unikalnych wartości, jakie występują we wskazanym polu w całej kolekcji — odpowiednik `SELECT DISTINCT miasto FROM uzytkownicy` w bazie relacyjnej.

Przykład:

```
[ 'Warszawa', 'Poznań', 'Kraków' ]
```

Z życia wzięte

> **Z ŻYCIA WZIĘTE**
>
> Kasia chce wycofać z katalogu jedną nieaktualną promocję i uruchamia updateMany z warunkiem, który miał dotyczyć tylko tej promocji, ale przez literówkę w nazwie pola pasuje do wszystkich produktów w kolekcji. W ciągu sekundy wszystkie produkty w sklepie B2B tracą pole cenaPromocyjna, co Marek zgłasza dopiero po południu, gdy klienci przestają widzieć promocje na stronie. Odtwarzamy poprawne wartości z ostatniego backupu opisanego na stronie Backup i odtwarzanie, a Kasia uczy się testować warunek find przed uruchomieniem update na całej kolekcji.

Częste błędy

Mylenie składni operatora zapytania z operatorem wyrażenia. `{ wiek: { $gt: 25 } }` działa w `find()`, ale wewnątrz potoku agregacji ten sam warunek zapisujemy jako `{ $gt: [ "$wiek", 25 ] }` — różnica opisana szerzej na stronie Operatory wyrażeń, a pomylenie tych dwóch form to jeden z najczęstszych błędów początkujących.

Wykonanie `deleteMany({})` bez zamierzonego filtra. Pusty filtr pasuje do wszystkich dokumentów kolekcji — literówka albo pominięty warunek potrafi w ten sposób wyczyścić całą kolekcję jednym poleceniem.

Użycie `replaceOne` tam, gdzie chodziło o częściową aktualizację. `replaceOne` nadpisuje cały dokument — pola nieobecne w podanym zastępniku znikają, co bywa zaskoczeniem dla kogoś przyzwyczajonego do `$set`.

Powiązane: Architektura MongoDB: dokument, kolekcja, baza, replica set · Aggregation pipeline · Operatory wyrażeń · Indeksy w MongoDB

Źródła: dokumentacja MongoDB, rozdział „CRUD Operations" i „Query and Projection Operators"

Revision #41
Created 2026-09-25 00:00:56 UTC by Tomasz Konieczny
Updated 2026-10-26 14:09:58 UTC by Tomasz Konieczny