Architektura MongoDB: dokument, kolekcja, baza, replica set

Poziom: podstawy **Dotyczy: MongoDB 5.0–8.0** **Czas: ~15 min**

Cztery pojęcia, na których stoi MongoDB — dokument, kolekcja, replica set i silnik przechowywania — oraz mapa tego, jak nazywają się te same rzeczy w bazie relacyjnej.

Skróty użyte na tej stronie (5)

Skrót	Co znaczy
B2B	business to business: sprzedaż i systemy skierowane do innych firm, nie do klientów indywidualnych
BSON	Binary JSON: binarny format zapisu dokumentów używany wewnętrznie przez MongoDB, w tym w archiwach mongodump
JSON	JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami
RPO	Recovery Point Objective: ile danych wolno stracić przy awarii
RTO	Recovery Time Objective: jak długo wolno nie działać przy awarii

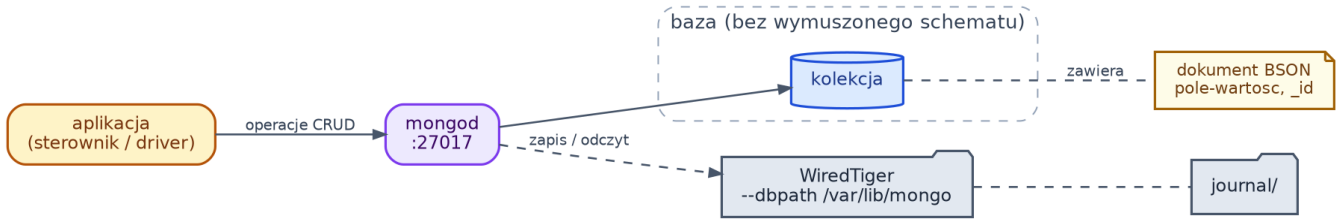
Z życia wzięte

Z ŻYCIA WZIĘTE

O czternastej w piątek Kasia wgrywa nową wersję sklepu B2B i dopisuje pole promocja do części dokumentów w kolekcji produkty na mongo01, nie przygotowując migracji danych historycznych, bo liczy na brak wymuszonego schematu. Strona listingu produktów zaczyna się wywalać dla starszych pozycji katalogu, bo kod frontendu zakłada, że pole promocja zawsze istnieje. Poprawka trafia do warstwy odczytu jako sprawdzenie istnienia pola przed jego użyciem, a nie jako migracja całej kolekcji naraz.

Po co to istnieje

Pojęcie: baza dokumentowa. MongoDB należy do rodziny baz dokumentowych, opisanej ogólnie na stronie Relacyjne, dokumentowe, kolumnowe, klucz-wartość: mapa rodzin — zamiast wierszy w sztywno zdefiniowanych tabelach, podstawową jednostką danych jest tu dokument: samodzielna, ustrukturyzowana porcja danych zapisana w formacie przypominającym JSON, mogąca zagnieżdżać w sobie kolejne obiekty i tablice.



Proces mongod przyjmuje operacje na kolekcji zlozonej z dokumentow BSON i zapisuje je przez WiredTiger na dysk (kliknięcie otwiera pełny rozmiar)

Pojęcie: BSON. Binary JSON — binarny format zapisu, w którym MongoDB faktycznie przechowuje dokumenty na dysku i przesyła je po sieci. Rozszerza składnię JSON o typy, których zwykły JSON nie ma, takie jak data, liczba 64-bitowa czy identyfikator binarny, i jest szybszy do parsowania niż tekstowy JSON, bo długości pól są zakodowane wprost w strumieniu bajtów zamiast wymagać skanowania znak po znaku.

Pojęcie: dokument. Pojedynczy rekord danych, odpowiednik wiersza w bazie relacyjnej, zapisany jako zestaw par pole-wartość. Każdy dokument ma pole `_id`, pełniące rolę klucza głównego, unikalne w obrębie kolekcji — jeśli go nie podamy przy wstawianiu, MongoDB wygeneruje je samo jako `ObjectId`.

Pojęcie: kolekcja. Zbiór dokumentów, odpowiednik tabeli w bazie relacyjnej. W odróżnieniu od tabeli kolekcja nie wymaga z góry zdefiniowanego schematu — nowy dokument może mieć pola, których nie ma żaden inny dokument w tej samej kolekcji.

Brak schematu — co to naprawdę znaczy

> **Pojęcie: schemat.** Ustalona z góry struktura danych: jakie pola ma mieć rekord i jakiego są typu. W silnikach relacyjnych schemat tabeli jest wymuszany przez silnik przy każdym zapisie, MongoDB go nie wymusza.

Brak wymuszonego schematu nie oznacza, że dane w kolekcji faktycznie są przypadkowe. W praktyce niemal każda kolekcja ma swój ustalony kształt, narzucany przez logikę aplikacji, która do niej zapisuje — po prostu ten kształt nie jest zapisany i pilnowany przez silnik bazy, tylko przez kod aplikacji albo, opcjonalnie, przez walidator schematu (`$jsonSchema`) skonfigurowany na poziomie kolekcji.

Konsekwencja praktyczna jest dwojaka. Z jednej strony dodanie nowego pola do dokumentów nie wymaga migracji całej kolekcji, tak jak `ALTER TABLE` w bazie relacyjnej opisany na stronie ALTER TABLE: co blokuje tabelę, a co nie — wystarczy zacząć je zapisywać w nowych dokumentach, a stare po prostu go nie mają. Z drugiej strony ta swoboda przenosi na aplikację odpowiedzialność za spójność, którą w bazie relacyjnej egzekwuje silnik przez `NOT NULL`, `CHECK` czy typy kolumn — kod odczytujący dokumenty musi sam radzić sobie z tym, że pole może nie istnieć, mieć nieoczekiwany typ, albo istnieć tylko w części rekordów historycznych.

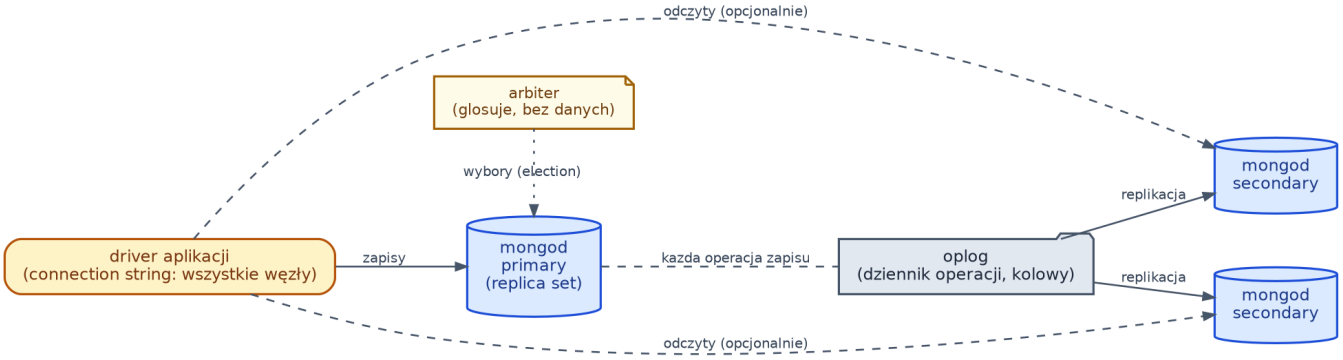
Procesy silnika

> **Pojęcie: mongod.** Główny proces serwera bazy danych — to on przyjmuje połączenia, wykonuje operacje na dokumentach i zarządza danymi na dysku. Pojedyncza instancja `mongod` nasłuchuje domyślnie na porcie `27017` i trzyma swoje pliki danych w katalogu wskazanym parametrem `--dbpath`.

> **Pojęcie: mongos.** Proces routera zapytań, obecny wyłącznie w konfiguracji z shardingiem — sam nie przechowuje żadnych danych, tylko przyjmuje zapytania od klientów i kieruje je do właściwego shardu (fragmentu danych) w klastrze. Aplikacja łączy się zawsze z `mongos`, nigdy bezpośrednio z pojedynczym `mongod` wchodzącym w skład shardowanego klastra.

Replica set

> **Pojęcie: replica set.** Grupa procesów `mongod` trzymających tę samą kopię danych, zapewniająca odporność na awarię pojedynczego serwera. W obrębie replica setu jeden węzeł pełni rolę `primary` i to on jako jedyny przyjmuje operacje zapisu; pozostałe węzły, `secondary`, replikują zapisy z primary asynchronicznie i mogą (jeśli aplikacja tak skonfiguruje odczyt) obsługiwać zapytania odczytu.



Primary przyjmuje zapisy i zapisuje je w oplogu, sekundary replikuja z niego, driver laczy sie z calym replica setem (kliknięcie otwiera pełny rozmiar)

Gdy primary przestaje odpowiadać, pozostałe węzły automatycznie przeprowadzają wybory (election) i jeden z secondary zostaje nowym primary — mechanizm ten wymaga nieparzystej liczby głosujących węzłów, żeby uniknąć remisu. Jeśli liczba pełnoprawnych węzłów danych wychodzi parzysta, dokłada się węzeł `arbiter` — proces uczestniczący w głosowaniu, ale nie trzymający kopii danych — wyłącznie po to, żeby zapewnić nieparzystość głosów.

Sharding w skrócie

> **Pojęcie: sharding.** Mechanizm poziomego podziału danych jednej kolekcji na kilka niezależnych serwerów (shardów), stosowany, gdy dane albo obciążenie zapytaniami przestają mieścić się na pojedynczym replica secie. Każdy shard jest sam w sobie osobnym replica setem, a to, który dokument trafia na który shard, decyduje wartość wybranego pola zwanego kluczem shardowania. Klaster shardowany dokłada dwa dodatkowe elementy nieobecne w pojedynczym replica secie: proces routera `mongos`, przez który przechodzą

wszystkie zapytania, oraz `config servers`, osobny replica set trzymający metadane o tym, który zakres kluczy shardowania leży na którym shardzie.

Analogia

Przejście od bazy relacyjnej do dokumentowej najłatwiej zobaczyć na przykładzie segregatora i teczki.

Baza relacyjna to segregator z gotowymi, jednakowymi formularzami — każda kartka (wiersz) ma z góry wydrukowane te same rubryki (kolumny), niezależnie od tego, czy dana sprawa akurat te rubryki wypełnia. Chcąc dopisać nową rubrykę do formularza, trzeba przedrukować cały nakład — to jest `ALTER TABLE` na dużej tabeli, operacja kosztowna właśnie dlatego, że dotyczy wszystkich kartek naraz. Baza dokumentowa to raczej teczka z osobnymi kartkami swobodnego formatu — jedna sprawa ma dołączony skan, inna ma dopisaną notatkę odręczną, a jeszcze inna nie ma nic poza samym wnioskiem. Nikt nie przedrukowuje całej teczki, żeby dodać nowy rodzaj załącznika do jednej sprawy.

Granica tej wygody jest taka sama jak granica samej teczki: łatwo w niej znaleźć kartkę po numerze sprawy przypiętym z góry (odpowiednik `_id`), trudniej szybko odpowiedzieć na pytanie w stylu „ile spraw ma załącznik typu X", bo nikt nie prowadził osobnego rejestru tego pola — dokładnie to zadanie przejmuje w MongoDB indeks założony świadomie na tym polu, opisany na stronie Indeksy w MongoDB.

Silnik przechowywania

Pojęcie: WiredTiger. Domyślny silnik przechowywania danych MongoDB od wersji 3.2 — odpowiada za to, jak dokumenty są faktycznie zapisywane i odczytywane z dysku, jak działa kompresja i jak przebiegają blokady na poziomie dokumentu. WiredTiger trzyma dane w katalogu wskazanym `--dbpath`, domyślnie `/var/lib/mongo` na dystrybucjach linuksowych, w plikach o rozszerzeniu `.wt` — po jednym na kolekcję i po jednym na każdy jej indeks, plus dziennik zapisu (`journal`) w podkatalogu `journal/`, zapewniający odtworzenie niezatwierdzonych jeszcze na dysk zmian po awarii.

Jak to działa: droga zapytania w replica secie

Klient aplikacyjny łączy się z całym replica setem naraz, podając w connection stringu adresy wszystkich jego węzłów, nie jednego wybranego. Sterownik (driver) MongoDB po stronie aplikacji sam ustala, który węzeł jest aktualnie `primary`, i kieruje do niego wszystkie operacje zapisu — próba zapisu wysłana wprost do węzła `secondary` kończy się odrzuceniem, bo `secondary` z definicji tylko odczytuje zmiany replikowane z `primary`, nigdy nie przyjmuje ich bezpośrednio od klienta.

Operacja zapisu na `primary` trafia najpierw do lokalnego zbioru danych węzła, a zaraz potem zostaje odnotowana w jego `oplogu`, opisanym na stronie Backup i odtwarzanie: mongodump i mongorestore — wewnętrznym, kołowym dzienniku operacji. Każdy `secondary` stale odczytuje ten dziennik z `primary` i powtarza u siebie te same operacje, w tej samej kolejności. To, jak bardzo świeże dane zwróci zapytanie odczytu skierowane do `secondary`, zależy więc od tego, jak daleko dany węzeł zdążył nadążyć za odczytem oplogu — opóźnienie to nazywa się replication lag i rośnie zwykle przy dużym obciążeniu zapisem albo przy słabszym łączu sieciowym do danego węzła.

Trwałość zapisu: write concern

> **Pojęcie: write concern.** Poziom potwierdzenia, jakiego aplikacja żąda od serwera, zanim operacja zapisu zostanie uznana za zakończoną powodzeniem. Domyślny poziom `w: 1` uznaje zapis za potwierdzony, gdy przyjmie go węzeł `primary`, jeszcze zanim zdąży go zreplikować do jakiegokolwiek `secondary` — najszybszy wariant, ale w razie awarii `primary` tuż po takim zapisie, zanim replikacja nadąży, dana zmiana może zostać utracona. Poziom `w: "majority"` każe czekać na potwierdzenie zapisu przez większość głosujących węzłów replica setu, kosztem nieco większego opóźnienia, za to z gwarancją, że zapis przetrwa utratę pojedynczego węzła, łącznie z samym `primary`.

Wybór między tymi poziomami jest właśnie tym, o czym mówi ogólna reguła RPO i RTO z fundamentów, opisana na stronie RPO i RTO: dwie liczby, od których zaczyna się każdy plan backupu — `w: 1` obniża opóźnienie zapisu, `w: "majority"` obniża ryzyko utraty danych przy awarii.

Mapa pojęć: relacyjne na dokumentowe

Relacyjnie	Dokumentowo (MongoDB)
tabela	kolekcja
wiersz	dokument
kolumna	pole
klucz główny	`_id`
indeks	indeks (`createIndex`, opisany na stronie Indeksy w MongoDB)
`JOIN`	`$lookup` w potoku agregacji, opisanym na stronie Aggregation pipeline
schemat tabeli	brak wymuszonego schematu (opcjonalny `$jsonSchema`)
transakcja wielotabelowa	transakcja wielodokumentowa (od MongoDB 4.0)

Częste błędy

Traktowanie braku schematu jako braku struktury. Kolekcja bez wymuszonego schematu wciąż ma swój faktyczny kształt narzucony przez aplikację — zmiana tego kształtu bez kontrolowanej migracji danych historycznych prowadzi do dokumentów, które trzeba obsługiwać dwoma różnymi ścieżkami kodu na raz.

Zakładanie, że `$lookup` działa jak `JOIN` bez konsekwencji wydajnościowych. `$lookup` w potoku agregacji jest operacją kosztowną, bo dla każdego dokumentu wejściowego przeszukuje osobno kolekcję docelową — model dokumentowy premiuje raczej zagnieżdżanie powiązanych danych wewnątrz jednego dokumentu niż łączenie wielu kolekcji przy każdym odczycie.

Powiązane: Relacyjne, dokumentowe, kolumnowe, klucz-wartość: mapa rodzin · mongosh: powłoka administracyjna · Aggregation pipeline · Indeksy w MongoDB

Źródła: notatki własne administratora MongoDB · dokumentacja MongoDB, rozdział „MongoDB Manual — Introduction" i „Replication"

Revision #42
Created 2026-09-25 00:00:54 UTC by Tomasz Konieczny
Updated 2026-10-26 14:09:55 UTC by Tomasz Konieczny