O'REILLY®

Wydanie III

Przewodnik po MongoDB

Wydajna i skalowalna baza danych

Shannon Bradshaw,
Eoin Brazil, Kristina Chodorow

Helion

Tytuł oryginału: MongoDB: The Definitive Guide: Powerful and Scalable Data Storage, 3rd Edition

Tłumaczenie: Wojciech Moch

ISBN: 978-83-283-6533-9

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/pmodb3.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/pmodb3
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 2.

Zaczynamy

MongoDB jest rozbudowanym systemem, który pozwala na łatwe rozpoczęcie pracy. W tym rozdziale przedstawimy zatem kilka podstawowych koncepcji istniejących w MongoDB.

- *Dokument* jest podstawową jednostką danych w MongoDB. Trochę przypomina wiersz w relacyjnym systemie zarządzania bazą danych (choć tutaj dokument jest bardziej ekspresyjny).
- O *kolekcji* można myśleć jak o tabeli z dynamiczną strukturą.
- Pojedyncza instancja systemu MongoDB może przechowywać kilka niezależnych *baz danych*, z których każda będzie składała się ze swoich własnych kolekcji.
- Każdy dokument ma specjalny klucz o nazwie `"_id"`, który jest unikalny w ramach danej kolekcji.
- MongoDB dostarczany jest wraz z prostym, ale też bardzo potężnym narzędziem o nazwie *powłoka mongo* (ang. *mongo shell*). W tej powłoce wbudowane są już funkcje umożliwiające administrowanie instancjami MongoDB oraz manipulowanie danymi przy użyciu języka zapytań MongoDB. Dostępny jest też w pełni funkcjonalny interpreter języka JavaScript pozwalający użytkownikom tworzyć i odczytywać własne skrypty o najróżniejszym przeznaczeniu.

Dokumenty

U podstawy MongoDB leży **dokument**: jest to uporządkowany zbiór kluczy z przypisanymi im wartościami. Reprezentacja dokumentu będzie różna w zależności od używanego języka programowania, ale w większości języków istnieją świetnie dopasowane do tego struktury danych, takie jak mapy, hashe lub słowniki. Na przykład w języku JavaScript dokumenty reprezentowane są przez obiekty:

```
{"greeting" : "Witaj, świecie!"}
```

Ten prosty dokument zawiera tylko jeden klucz — `"greeting"`, którego wartość to `"Witaj, świecie!"`. Większość dokumentów jest bardziej rozbudowana niż ten prosty przykład. Bardzo często zawierają one wiele różnych par klucz – wartość.

```
{"greeting" : "Witaj, świecie!", "views" : 3}
```

Jak widać, wartości zapisane w dokumencie mają konkretne typy. Mogą one przyjmować postać jednego z kilku różnych typów danych (a nawet mogą być całymi zagnieżdżonymi dokumentami

— zajrzyj do punktu „Zagnieżdżone dokumenty"). W tym przykładzie klucz `"greeting"` ma wartość ciągu znaków, natomiast klucz `"views"` przechowuje wartość całkowitą.

Klucze w każdym dokumencie są ciągami znaków. W kluczu można użyć dowolnego znaku UTF-8, z kilkoma ważnymi wyjątkami:

- W kluczu nie można umieścić znaku zerowego (o kodzie `\0`), ponieważ ten znak używany jest do oznaczenia końca klucza.
- Znaki kropki (`.`) i dolara (`$`) mają specjalne właściwości i dlatego powinny być używane tylko w określonych okolicznościach, o których opowiemy w dalszej części rozdziału. W większości przypadków należy traktować je jak znaki zastrzeżone. Jeżeli użyjemy ich w niewłaściwy sposób, to sterowniki będą zgłaszały błędy.

MongoDB jest systemem rozróżniającym typy i wielkości liter. Na przykład poniżej widoczne są dwa różne dokumenty:

```
{"count" : 5}
{"count" : "5"}
```

Dokumenty również są różne:

```
{"count" : 5}
{"Count" : 5}
```

Ostatnią ważną rzeczą, o której musimy wspomnieć, jest to, że dokumenty w MongoDB nie mogą zawierać duplikatów kluczy. Na przykład poniższy dokument nie jest prawidłowy:

```
{"greeting" : "Witaj, świecie!", "greeting" : "Witaj, MongoDB!"}
```

Kolekcje

Kolekcja jest grupą dokumentów. Jeżeli dokument w MongoDB jest odpowiednikiem wiersza w relacyjnej bazie danych, to kolekcja może być traktowana jak odpowiednik tabeli.

Dynamiczne schematy

Kolekcje mają **dynamiczne schematy** (ang. *dynamic schemas*). Oznacza to, że dokumenty zgromadzone w jednej kolekcji mogą mieć dowolną liczbę różnych „kształtów". Na przykład w jednej kolekcji mogą znaleźć się oba przedstawione niżej dokumenty:

```
{"greeting" : "Witaj, świecie!", "views": 3}
{"signoff": "Dobranoc i powodzenia."}
```

Zauważ, że powyższe dokumenty mają różne klucze, inną liczbę kluczy i przechowują wartości o różnych typach. Ze względu na to, że dowolny dokument może zostać umieszczony w dowolnej kolekcji, powstaje pytanie: „Po co nam w ogóle niezależne kolekcje"? Skoro nie istnieje wymóg stosowania jednolitego schematu dokumentów, to dlaczego mielibyśmy stosować jakiekolwiek dodatkowe kolekcje? Można tutaj podać kilka dobrych powodów:

- Umieszczanie w jednej kolekcji kilku rodzajów dokumentów może okazać się koszmarem zarówno dla administratorów, jak i dla programistów. Programiści muszą się upewniać, że każde zapytanie

zwraca wyłącznie dokumenty pasujące do określonego schematu albo tworzyć taki kod aplikacji, który poradzi sobie w przypadku, gdy zapytanie zwróci dokumenty o różnych strukturach. Jeżeli wysyłamy zapytanie pobierające posty umieszczone na blogu, to bardzo kłopotliwe byłoby odsiewanie dokumentów zawierających dane autorów.

- Znacznie szybciej można pobrać listę kolekcji niż wydobyć listę typów dokumentów zapisanych w kolekcji. Na przykład, jeżeli w każdym dokumencie mielibyśmy pole `"typ"`, które określałoby, czy dany dokument jest typu `"przegląd"`, `"całość"` lub `"kawałeczek"`, to wyszukiwanie tych trzech wartości w całej kolekcji byłoby znacznie wolniejsze, niż gdybyśmy mieli trzy kolekcje i mogli wysłać zapytanie do jednej z nich.
- Grupowanie dokumentów tego samego typu w ramach jednej kolekcji pozwala na uzyskanie lokalności danych. Pobranie kilku postów na blogu z kolekcji przechowującej wyłącznie takie posty będzie wymagało mniej operacji poszukiwania na dysku niż próby pobrania tych samych postów z kolekcji przechowującej posty i dane autorów.
- Tworząc indeksy, zaczynamy narzucać pewną strukturę naszych dokumentów. (Jest to szczególnie prawdziwe w przypadku stosowania indeksów unikalnych). Takie indeksy są definiowane dla każdej kolekcji z osobna. Umieszczając w kolekcji wyłącznie dokumenty jednego typu, możemy zwiększyć skuteczność jej indeksowania.

To bardzo dobre powody tworzenia schematów i grupowania ze sobą dokumentów o podobnym typie. Choć nie jest to domyślnie wymagane, to jednak dobrą praktyką jest definiowanie schematów w aplikacji, co może być wymuszane przez funkcje kontroli poprawności wbudowane w MongoDB, a także przez biblioteki odwzorowania obiektów na dokumenty, które są dostępne w wielu językach programowania.

Nazewnictwo

Kolekcja jest definiowana za pomocą nazwy. Nazwy kolekcji mogą być dowolnymi ciągami znaków UTF-8, z kilkoma ważnymi ograniczeniami:

- Pusty ciąg znaków (`""`) nie jest poprawną nazwą kolekcji.
- Nazwy kolekcji nie mogą zawierać znaku zerowego (`\0`), ponieważ jest on używany do wyznaczania końca nazwy kolekcji.
- Nie należy tworzyć kolekcji, których nazwa zaczyna się od słowa `system.`, ponieważ ten przedrostek jest zarezerwowany dla kolekcji wewnętrznych. Na przykład kolekcja `system.users` przechowuje dane użytkowników bazy danych, natomiast kolekcja `system.namespaces` przechowuje informacje o wszystkich kolekcjach danej bazy danych.
- Nazwy kolekcji tworzonych przez użytkowników nie powinny zawierać zarezerwowanego znaku dolara (`$`). Niektóre sterowniki obsługujące te bazy danych pozwalają na stosowanie znaku dolara w nazwie kolekcji, ponieważ niektóre systemowe kolekcje zawierają takie znaki, ale mimo to nie należy używać znaków dolara, chyba że chcesz uzyskać dostęp do jednej z takich kolekcji.

Podkolekcje

Jedną z konwencji organizowania kolekcji jest używanie przestrzeni nazw dla podkolekcji, w których poszczególne nazwy oddzielane są znakiem kropki (`.`). Na przykład aplikacja obsługująca blog może korzystać z kolekcji o nazwie `blog.posts` do przechowywania postów oraz kolekcji `blog.authors` do przechowywania danych autorów. Takie nazewnictwo ma jedynie poprawić organizację kolekcji. Nie istnieje żaden związek pomiędzy kolekcją `blog` (ta może nawet nie istnieć) a jej „dziećmi".

Choć podkolekcje nie mają żadnych specjalnych właściwości, to jednak są naprawdę przydatne i są wykorzystywane przez wiele narzędzi dla MongoDB. Na przykład:

- GridFS jest protokołem do przechowywania wielkich plików, który używa podkolekcji do zapisywania metadanych pliku niezależnie od części jego treści. (Więcej informacji na temat GridFS znajdziesz w rozdziale 6.).
- Większość sterowników dodaje składniowe ułatwienia w dostępie do podkolekcji wybranej kolekcji. Na przykład w powłoce bazy danych wpisanie nazwy `db.blog` zwróci kolekcję `blog`, natomiast nazwa `db.blog.posts` zwróci kolekcję `blog.posts`.

Podkolekcje w MongoDB są świetną metodą organizowania danych w wielu różnych przypadkach.

Bazy danych

Oprócz grupowania dokumentów w kolekcje MongoDB grupuje też kolekcje w ramach **baz danych**. Jedna instancja MongoDB może przechowywać wiele różnych baz danych, z których każda może być zbiorem różnych kolekcji. Ważna reguła mówi, żeby wszystkie dane związane z jedną aplikacją przechowywać w tej samej bazie danych. Osobne bazy danych przydają się wtedy, gdy na jednym serwerze MongoDB przechowujemy dane dla kilku aplikacji lub użytkowników.

Podobnie jak kolekcje, bazy danych również są identyfikowane przez nazwę. Nazwy baz danych mogą być dowolnymi ciągami znaków UTF-8, z kilkoma ważnymi ograniczeniami:

- Pusty ciąg znaków (`""`) nie jest poprawną nazwą bazy danych.
- Nazwa bazy danych nie może zawierać żadnego z tych znaków: /, \, ., ", *, <, >, :, |, ?, $, pojedyncza spacja oraz `\0` (znak zerowy). Najprościej mówiąc, lepiej ograniczyć się do znaków alfanumerycznych zbioru ASCII.
- W nazwach baz danych rozróżniamy wielkość liter.
- Wielkość nazwy bazy danych nie może przekroczyć 64 bajtów.

Zanim pojawił się mechanizm przechowywania danych WiredTiger, nazwy baz danych przekładały się bezpośrednio na nazwy plików na dysku. Dzisiaj już tego nie zobaczymy. To jednak tłumaczy, dlaczego do dzisiaj istnieje jeszcze wiele ograniczeń pochodzących z dawnych czasów.

Istnieje też kilka zarezerwowanych nazw baz danych, do których można uzyskać dostęp, a które mają specjalną semantykę. Do tych nazw należą:

admin

Baza danych *admin* pełni ważną rolę w procesie uwierzytelniania i autoryzacji. Dodatkowo niektóre działania administracyjne również wymagają uzyskania dostępu do tej bazy danych. Więcej informacji na temat bazy danych *admin* znajdziesz w rozdziale 19.

local

W ten bazie danych przechowywane są dane opisujące sam serwer. W przypadku zbiorów replik baza danych *local* przechowuje też informacje na temat procesu replikacji. Ważne jest to, że baza danych *local* nigdy nie podlega replikacji. Więcej informacji na temat procesu replikacji oraz bazy danych *local* znajdziesz w rozdziale 10.

config

We współdzielonych klastrach MongoDB (będziemy o nich mówić w rozdziale 14.) baza danych *config* jest używana do przechowywania informacji o poszczególnych shardach.

Łącząc nazwę bazy danych z nazwą kolekcji przechowywanej w tej bazie, uzyskujemy pełną nazwę tej kolekcji, która jest też nazywana **przestrzenią nazw** (ang. *namespace*). Na przykład, jeżeli w bazie danych *cms* utrzymujesz kolekcję `blog.posts`, to przestrzenią nazw tej kolekcji będzie `cms.blog.posts`. Przestrzeni nazw dotyczy ograniczenie wielkości do 120 bajtów, a w praktyce nie powinny one przekraczać wielkości 100 bajtów. Więcej informacji na temat przestrzeni nazw i wewnętrznej reprezentacji kolekcji w MongoDB znajdziesz w dodatku B.

Pobieranie i uruchamianie MongoDB

Aby uruchomić serwer, wywołaj polecenie `mongod` w używanym przez siebie uniksowym środowisku wiersza poleceń:

```
$ mongod
2016-04-27T22:15:55.871-0400 I CONTROL [initandlisten] MongoDB starting :
pid=8680 port=27017 dbpath=/data/db 64-bit host=morty
2016-04-27T22:15:55.872-0400 I CONTROL [initandlisten] db version v4.2.0
2016-04-27T22:15:55.872-0400 I CONTROL [initandlisten] git version:
34e65e5383f7ea1726332cb175b73077ec4a1b02
2016-04-27T22:15:55.872-0400 I CONTROL [initandlisten] allocator: system
2016-04-27T22:15:55.872-0400 I CONTROL [initandlisten] modules: none
2016-04-27T22:15:55.872-0400 I CONTROL [initandlisten] build environment:
2016-04-27T22:15:55.872-0400 I CONTROL [initandlisten] distarch: x86_64
2016-04-27T22:15:55.872-0400 I CONTROL [initandlisten] target_arch: x86_64
2016-04-27T22:15:55.872-0400 I CONTROL [initandlisten] options: {}
2016-04-27T22:15:55.889-0400 I JOURNAL [initandlisten]
journal dir=/data/db/journal
2016-04-27T22:15:55.889-0400 I JOURNAL [initandlisten] recover :
no journal files
present, no recovery needed
2016-04-27T22:15:55.909-0400 I JOURNAL [durability] Durability thread started
2016-04-27T22:15:55.909-0400 I JOURNAL [journal writer] Journal writer thread
started
2016-04-27T22:15:55.909-0400 I CONTROL [initandlisten]
2016-04-27T22:15:56.777-0400 I NETWORK [HostnameCanonicalizationWorker]
```

```
Starting hostname canonicalization worker
2016-04-27T22:15:56.778-0400 I FTDC [initandlisten] Initializing full-time
diagnostic data capture with directory '/data/db/diagnostic.data'
2016-04-27T22:15:56.779-0400 I NETWORK [initandlisten] waiting for connections
on port 27017
```

Jeżeli pracujesz w systemie Windows, uruchom to polecenie:

```
> mongod.exe
```

Dokładne informacje na temat instalowania MongoDB w swoim systemie znajdziesz w dodatku A albo w specjalnym poradniku instalowania (*https://docs.mongodb.com/manual/installation/*) będącym częścią dokumentacji MongoDB.

Po uruchomieniu bez żadnych argumentów program `mongod` będzie korzystał z domyślnego katalogu */data/db* (a w systemie Windows — *\data\db* na aktualnym wolumenie). Jeżeli katalog z danymi nie istnieje albo nie pozwala na zapis, to serwer się nie uruchomi. Z tego powodu bardzo ważne jest utworzenie tego katalogu (na przykład poleceniem `mkdir -p /data/db/`) i nadanie mu uprawnień do zapisu jeszcze przed próbą uruchomienia MongoDB.

Podczas uruchamiania serwer wypisze krótką informację o swojej wersji i używanym systemie operacyjnym, a następnie będzie oczekiwał na połączenia. Domyślnie MongoDB nasłuchuje połączeń na porcie 27017. Serwer się nie uruchomi, jeżeli ten port nie będzie dla niego dostępny. Najczęstszym powodem jest tu działająca już inna instancja serwera MongoDB.

Zawsze należy odpowiednio zabezpieczać instancje serwera *mongod*. Więcej informacji na temat zabezpieczenia MongoDB znajdziesz w rozdziale 19.

Możesz też bezpiecznie zatrzymać serwer *mongod*, naciskając klawisze *Ctrl+C* w oknie wiersza poleceń, w którym został on wcześniej uruchomiony.

Więcej informacji na temat uruchamiania i zatrzymywania MongoDB znajdziesz w rozdziale 21.

Wprowadzenie do powłoki MongoDB

MongoDB dostarczane jest razem z powłoką JavaScriptu, która umożliwia interakcję z instancjami MongoDB z poziomu wiersza poleceń. Ta powłoka bardzo przydaje się do wykonywania zadań administracyjnych, kontrolowania działających instancji oraz do eksperymentowania. Powłoka *mongo* jest bardzo ważnym narzędziem podczas korzystania z baz danych MongoDB. W dalszej części książki będziemy z niej bardzo intensywnie korzystać.

Uruchamianie powłoki

Aby uruchomić powłokę, znajdź i uruchom plik wykonywalny *mongo*:

```
$ mongo
MongoDB shell version: 4.2.0
connecting to: test
>
```

Powłoka automatycznie próbuje połączyć się z serwerem MongoDB działającym na tym samym komputerze, dlatego upewnij się, że serwer *mongod* został już wcześniej uruchomiony.

Powłoka jest pełnoprawnym interpreterem języka JavaScript, zdolnym do uruchomienia dowolnych programów w języku JavaScript. Aby się o tym przekonać, możesz wykonać kilka prostych obliczeń:

```
> x = 200;
200
> x / 5;
40
```

Można też korzystać ze wszystkich standardowych bibliotek języka JavaScript:

```
> Math.sin(Math.PI / 2);
1
> new Date("2019/1/1");
ISODate("2019-01-01T05:00:00Z")
> "Witaj, świecie!".replace("świecie", "MongoDB");
Witaj, MongoDB!
```

Możemy też zdefiniować i wywoływać własne funkcje:

```
> function factorial(n) {
... if (n <= 1) return 1;
... return n * factorial(n - 1);
... }
> factorial(5);
120
```

Zauważ, że możliwe jest wprowadzanie poleceń wielowierszowych. Po naciśnięciu klawisza *Enter* powłoka sama wykrywa, czy instrukcja JavaScriptu jest już zakończona. Jeżeli tak nie jest, to powłoka pozwala nam kontynuować wpisywanie od następnego wiersza. Trzykrotne naciśnięcie klawisza *Enter* powoduje anulowanie niedokończonej instrukcji i powrót do standardowego znaku zachęty (>).

Klient MongoDB

Co prawda możliwość wykonywania dowolnego kodu języka JavaScript jest bardzo przydatna, ale prawdziwa moc powłoki wynika z faktu, że jest ona niezależnym klientem MongoDB. Po uruchomieniu powłoka łączy się z bazą danych *test* istniejącą na serwerze MongoDB i przypisuje to połączenie do globalnej zmiennej o nazwie `db`. Ta zmienna staje się podstawowym punktem dostępu do serwera MongoDB z poziomu powłoki.

Aby zobaczyć nazwę bazy danych, z jaką aktualnie jest powiązana zmienna `db`, należy wpisać w powłoce nazwę tej zmiennej i nacisnąć klawisz *Enter*:

```
> db
test
```

W powłoce dostępnych jest kilka rozszerzeń, które nie są co prawda zgodne ze składnią języka JavaScript, ale zostały zaimplementowane, ponieważ są to operacje dobrze znane użytkownikom powłok baz danych SQL. Takie dodatki nie wprowadzają żadnych nowych funkcji, a są jedynie pewnymi ułatwieniami składniowymi. Na przykład jedną z najważniejszych operacji jest wybieranie bazy danych, z którą będziemy pracować:

```
> use video
switched to db video
```

Jeżeli teraz przyjrzysz się zmiennej `db`, to zauważysz, że odwołuje się ona do bazy danych *video*:

```
> db
video
```

Ze względu na to, że mamy do czynienia z powłoką języka JavaScript, wpisanie nazwy zmiennej sprawia, że jest ona traktowana jak wyrażenie. W efekcie wypisywana jest wartość zmiennej (w tym przypadku jest to nazwa bazy danych).

Za pośrednictwem zmiennej `db` można uzyskać dostęp do kolekcji. Na przykład zapis:

```
> db.movies
```

zwróci nam kolekcję `movies` znajdującą się w aktualnej bazie danych. Skoro możemy już w powłoce uzyskać dostęp do kolekcji, to możemy też wykonać niemalże dowolną operację na bazie danych.

Podstawowe operacje w powłoce

Do przeglądania danych i manipulowania nimi w powłoce możemy wykorzystywać cztery podstawowe operacje CRUD: tworzenia (ang. create), odczytu (ang. read), aktualizacji (ang. update) i usuwania (ang. delete).

Tworzenie

Funkcja `insertOne` dodaje do kolekcji jeden dokument. Załóżmy, że w naszej bazie chcemy zapisać dane filmu. Najpierw musimy utworzyć zmienną lokalną o nazwie `movie`, która w języku JavaScript będzie reprezentować nasz dokument. Sam dokument będzie przechowywał klucze `"title"` (tytuł filmu), `"director"` (reżyser) oraz `"year"` (rok pojawienia się filmu):

```
> movie = {"title" : "Star Wars: Epizod IV – Nowa nadzieja",
... "director" : "George Lucas",
... "year" : 1977}
{
    "title" : "Star Wars: Epizod IV – Nowa nadzieja",
    "director" : "George Lucas",
    "year" : 1977
}
```

Ten obiekt jest poprawnym dokumentem MongoDB, który możemy zapisać w kolekcji `movies`, używając do tego metody `insertOne`:

```
> db.movies.insertOne(movie)
{
    "acknowledged" : true,
    "insertedId" : ObjectId("5721794b349c32b32a012b11")
}
```

Dane filmu zostały zapisane w bazie danych. Możemy to potwierdzić, wywołując w kolekcji metodę `find`:

```
> db.movies.find().pretty()
{
    "_id" : ObjectId("5721794b349c32b32a012b11"),
    "title" : "Star Wars: Epizod IV – Nowa nadzieja",
    "director" : "George Lucas",
    "year" : 1977
}
```

Widać tutaj, że wszystkie wprowadzone przez nas pary klucz – wartość zostały zapisane bez zmian, a dodatkowo w dokumencie znalazł się klucz `"_id"`. Powód niespodziewanego pojawienia się klucza `"_id"` zostanie wyjaśniony pod koniec tego rozdziału.

Odczyt

Do odczytywania danych z kolekcji można używać funkcji `find` lub `findOne`. Jeżeli chcemy otrzymać tylko jeden dokument z kolekcji, to możemy użyć funkcji `findOne`:

```
> db.movies.findOne()
{
    "_id" : ObjectId("5721794b349c32b32a012b11"),
    "title" : "Star Wars: Epizod IV – Nowa nadzieja",
    "director" : "George Lucas",
    "year" : 1977
}
```

Funkcjom `find` i `findOne` można też przekazywać kryteria wyszukiwania w postaci dokumentu zapytania. To zawęzi uzyskaną odpowiedź do dokumentów pasujących do warunków zapytania. Powłoka wyświetla automatycznie maksymalnie 20 dokumentów pasujących do kryteriów podanych funkcji `find`, ale można pobrać ich dowolną ilość. (Więcej informacji na temat tworzenia zapytań znajdziesz w rozdziale 4.).

Aktualizowanie

Jeżeli chcielibyśmy poprawić zawartość naszego dokumentu, to można użyć funkcji `updateOne`. Funkcja ta pobiera (przynajmniej) dwa parametry: pierwszym jest warunek pozwalający wyszukać dokument do aktualizacji, a drugim jest dokument opisujący zmiany, jakie należy wprowadzić. Załóżmy, że chcemy pozwolić na podawanie recenzji filmu, dla którego już wcześniej utworzyliśmy dokument. Musimy zatem dodać tablicę z recenzjami, która stanie się nowym kluczem w naszym dokumencie.

Aby wykonać taką aktualizację, musimy wykorzystać operator aktualizowania — `set`:

```
> db.movies.updateOne({title : "Star Wars: Epizod IV – Nowa nadzieja"},
... {$set : {reviews: []}})
WriteResult({"nMatched": 1, "nUpserted": 0, "nModified": 1})
```

Od teraz nasz dokument ma też klucz o nazwie `"reviews"`. Jeżeli ponownie wywołamy funkcję `find`, to w wyświetlonym dokumencie zauważymy nowy klucz:

```
> db.movies.find().pretty()
{
    "_id" : ObjectId("5721794b349c32b32a012b11"),
    "title" : "Star Wars: Epizod IV – Nowa nadzieja",
    "director" : "George Lucas",
    "year" : 1977,
    "reviews" : [ ]
}
```

Więcej informacji na temat aktualizowania dokumentów znajdziesz w podrozdziale „Aktualizowanie dokumentów”.

Usuwanie

Funkcje `deleteOne` i `deleteMany` trwale usuwają dokumenty z bazy danych. Obie metody pobierają w parametrze dokument filtra określający kryteria usuwania. Na przykład to polecenie usunie dokument filmu, który przed chwilą utworzyliśmy:

```
> db.movies.deleteOne({title : "Star Wars: Epizod IV – Nowa nadzieja"})
```

Za pomocą funkcji `deleteMany` można usunąć wszystkie dokumenty pasujące do podanego filtra.

Typy danych

Na początku tego rozdziału przedstawiliśmy podstawowe informacje o tym, czym właściwie jest dokument. Teraz gdy masz już uruchomiony serwer MongoDB i możesz wypróbować w powłoce różne rzeczy, w tym podrozdziale przyjrzymy się kilku szczegółom. MongoDB obsługuje ogromną liczbę różnych typów danych, które można zapisać jako wartości w dokumentach. W tym podrozdziale opiszemy wszystkie obsługiwane typy danych.

Proste typy danych

Dokumenty w bazach danych MongoDB można traktować jak elementy „podobne do JSON”, ponieważ w ogólnej koncepcji są one zbliżone do obiektów języka JavaScript. Format JSON (*https://www.json.org/json-pl.html*) jest prostym sposobem reprezentowania danych. Jego specyfikacja mieści się w jednym akapicie tekstu (przekonasz się o tym na podanej stronie) i wymienia jedynie część typów danych. Pod wieloma względami to bardzo dobre rozwiązanie: łatwo je poznać, takie dokumenty łatwo się parsuje, a jeszcze łatwiej zapamiętuje. Z drugiej strony możliwości wyrażania danych w formacie JSON są bardzo ograniczone, ponieważ pozwala on na stosowanie wyłącznie typów `null`, logicznych, liczbowych, ciągów znaków, tablic i obiektów.

Choć już tylko te typy pozwalają na zapisanie imponująco zróżnicowanych informacji, to jednak dodano też obsługę kilku innych typów, bardzo ważnych podczas pracy z większością aplikacji współpracujących z bazami danych. Na przykład w formacie JSON nie ma typu daty, co sprawia, że praca z datami staje się jeszcze bardziej skomplikowana niż zazwyczaj. Istnieje wprawdzie typ numeryczny, ale tylko jeden. Nie ma rozróżnienia między liczbami stało- i zmiennoprzecinkowymi, nie wspominając nawet o rozróżnianiu liczb 32- i 64-bitowych. Nie istnieje też metoda reprezentowania innych często używanych typów, takich jak wyrażenia regularne lub funkcje.

MongoDB udostępnia też obsługę kilku dodatkowych typów danych, zachowując właściwą dla formatu JSON naturę par klucz – wartość. Dokładny sposób reprezentowania poszczególnych typów danych uzależniony jest od używanego języka programowania. Poniżej przedstawiamy listę często używanych typów i sposobów ich reprezentacji w ramach dokumentu obsługiwanego w powłoce. Do najczęściej spotykanych typów należą:

Null

Typ `null` może być używany do reprezentowania zarówno wartości `null`, jak i do oznaczania nieistniejącego pola:

```
{"x" : null}
```

Logiczny (ang. boolean*)*

Istnieje też typ logiczny, pozwalający na stosowanie wartości prawdy (ang. `true`) lub fałszu (ang. `false`):

```
{"x" : true}
```

Liczbowy (ang. number*)*

Powłoka domyślnie stosuje 64-bitowe liczby zmiennoprzecinkowe, w związku z tym obie poniższe liczby są dla niej czymś „normalnym”:

```
{"x" : 3.14}
{"x" : 3}
```

W przypadku stosowania liczb całkowitych można użyć klas `NumberInt` lub `NumberLong`, które reprezentują liczy całkowite o wielkości 4 lub 8 bajtów.

```
{"x" : NumberInt("3")}
{"x" : NumberLong("3")}
```

Ciąg znaków (ang. string*)*

Za pomocą typu string można przedstawić dowolny ciąg znaków UTF-8:

```
{"x" : "foobar"}
```

Data (ang. date*)*

MongoDB przechowuje daty w postaci 64-bitowych liczb całkowitych odpowiadających liczbie milisekund upływających od początku epoki Uniksa (1 stycznia 1970 roku). Informacja o strefie czasowej nie jest przechowywana.

```
{"x" : new Date()}
```

Wyrażenia regularne (ang. regular expression*)*

W zapytaniach można stosować wyrażenia regularne zgodne ze składnią wyrażeń regularnych języka JavaScript.

```
{"x" : /foobar/i}
```

Tablice (ang. arrays*)*

Zbiór lub lista wartości może zostać zapisana w postaci tablicy:

```
{"x" : ["a", "b", "c"]}
```

Dokument zagnieżdżony (ang. embedded document*)*

Dokumenty mogą zawierać w sobie całe dokumenty zagnieżdżone jako wartości w dokumencie nadrzędnym:

```
{"x" : {"foo" : "bar"}}
```

Identyfikator (ang. Object ID*)*

Identyfikator obiektu jest 12-bajtowym identyfikatorem poszczególnych dokumentów:

```
{"x" : ObjectId()}
```

Więcej informacji na ten temat znajdziesz w punkcie „Klucz _id i typ ObjectId”.

Istnieje też kilka rzadziej używanych typów, które mimo wszystko czasem się przydają:

Dane binarne (ang. binary data*)*

Dane binarne są ciągiem bajtów o dowolnej wartości. Tym typem danych nie można się posługiwać z poziomu powłoki. Binarne dane są jedynym sposobem zapisania w bazie danych ciągów znaków innych niż UTF-8.

Kod (ang. code*)*

MongoDB pozwala też na zapisywanie w dokumentach dowolnych zapytań języka JavaScript:

```
{"x" : function() { /* ... */ }}
```

Poza tym istnieje jeszcze kilka typów wykorzystywanych przede wszystkim przez sam system MongoDB (albo które zostały zastąpione innymi typami). Będziemy o nich mówić w razie potrzeby w dalszej części książki.

Więcej informacji na temat formatu danych używanego przez MongoDB znajdziesz w dodatku B.

Daty

W języku JavaScript klasa `Date` służy do przechowywania informacji o dacie z MongoDB. Tworząc nowy obiekt typu `Date`, trzeba zawsze użyć wywołania `new Date()`, a nie wywołania `Date()`. Wywołanie konstruktora jako funkcji (bez użycia słowa kluczowego `new`) powoduje zwrócenie ciągu znaków opisującego datę, a nie sam obiekt typu `Date`. MongoDB nie ma na to żadnego wpływu, tak działa już sam język JavaScript. Jeżeli nie zachowasz ostrożności przy wywoływaniu konstruktora klasy `Date`, uzyskasz w kodzie paskudną mieszankę ciągów znaków i obiektów. Ciągi znaków nie dają się porównywać z obiektami daty, co z kolei powoduje różne problemy przy usuwaniu, aktualizowaniu, odczytywaniu… właściwie przy każdej operacji, o jakiej sobie pomyślisz.

Pełny opis klasy `Date` z języka JavaScript wraz z listą poprawnych wywołań konstruktora można znaleźć w sekcji 15.9 specyfikacji ECMAScript (*https://www.ecma-international.org/*).

W powłoce daty są wyświetlane z uwzględnieniem ustawień lokalnej strefy czasowej. W samej bazie danych daty są jednak zapisywane jako liczba milisekund od początku epoki Uniksa, co oznacza, że nie mają dodanej informacji o strefie czasowej. (Informację o strefie czasowej można oczywiście zapisać jako wartość innego klucza).

Tablice

Tablice są wartościami, które można wymiennie stosować do przechowywania danych uporządkowanych (jakby były one listami, stosami lub kolejkami) lub danych nieuporządkowanych (jakby były prostymi zbiorami).

W poniższym dokumencie klucz `"things"` ma wartość będącą tablicą:

```
{"things" : ["pi", 3.14]}
```

Jak można zauważyć w tym przykładzie, tablice mogą przechowywać wartości o różnych typach danych (w tym przypadku jest to ciąg znaków i liczba zmiennoprzecinkowa). Co ciekawe, wartości w tablicy mogą mieć dowolny z typów danych obsługiwanych przez normalne pary klucz – wartość. Mogą być nawet zagnieżdżonymi tablicami.

Jedną z miłych cech tablic umieszczanych w dokumentach jest to, że MongoDB „zna" ich strukturę i wie, jak należy z nich korzystać, aby wykonywać różne operacje na poszczególnych wartościach. Pozwala to na tworzenie zapytań dotyczących tablic oraz na tworzenie indeksów korzystających z ich zawartości. Rozwijając poprzedni przykład, można powiedzieć, że MongoDB umożliwia wyszukiwanie wszystkich dokumentów, w których wartość `3.14` będzie elementem tablicy o nazwie `"things"`. Jeżeli takie zapytanie będzie często używane, to można dodatkowo utworzyć indeks według klucza `"things"`, aby przyspieszyć proces wyszukiwania.

MongoDB pozwala też na stosowanie atomowych aktualizacji modyfikujących zawartość tablic, na przykład polegających na wymianie w tablicy wartości `"pi"` na wartość `pi`. W dalszej części zobaczymy więcej przykładów operacji tego rodzaju.

Zagnieżdżone dokumenty

Dokument może zostać użyty jako wartość danego klucza. Nazywa się to **zagnieżdżaniem dokumentów** (ang. *embedded document*). Zagnieżdżania dokumentów można używać do bardziej naturalnego organizowania danych, na które nie pozwalałaby płaska struktura par klucz – wartość.

Na przykład, jeżeli mamy dokument opisujący osobę i chcemy w nim zapisać jej adres zamieszkania, to możemy zastosować zagnieżdżony dokument o nazwie `"address"`:

```
{
    "name" : "Jan Kowalski",
    "address" : {
        "street" : "Parkowa 123",
        "city" : "Miasteczko",
```

```
            "state" : "Opolskie"
        }
    }
```

W tym przykładzie wartość klucza `"address"` jest zagnieżdżonym dokumentem zawierającym własne pary klucz – wartość o nazwach `"street"`, `"city"` i `"state"`.

Podobnie jak w przypadku tablic, tutaj MongoDB również „zna” strukturę zagnieżdżonego dokumentu i pozwala na skorzystanie z niej podczas tworzenia indeksów, wykonywania zapytań i aktualizowania.

Na temat projektowania schematów będziemy mówić później, ale nawet w tym prostym przykładzie można zauważyć, że zagnieżdżone dokumenty mogą zmienić nasz sposób pracy z danymi. W relacyjnej bazie danych powyższy dokument zostałby najprawdopodobniej zamodelowany jako dwa osobne wiersze w dwóch niezależnych tabelach (o nazwach `people` i `addresses`). W MongoDB możemy umieścić dokument `"address"` bezpośrednio w dokumencie `"person"`. Jak widać, poprawnie zastosowane dokumenty zagnieżdżone mogą stać się metodą bardziej naturalnej reprezentacji informacji.

Z drugiej strony z tego właśnie powodu w bazach danych MongoDB może pojawiać się znacznie więcej powtarzających się informacji. Załóżmy, że dokument `"addresses"` byłby osobną tabelą w relacyjnej bazie danych, a my musielibyśmy poprawić mały błąd w adresie. Po złączeniu tabel `adresses` i `people` taką poprawkę otrzymalibyśmy dla wszystkich powiązanych osób. W MongoDB tę samą poprawkę musielibyśmy wprowadzić osobno w dokumencie każdej z tych osób.

Klucz _id i typ ObjectId

Każdy dokument zapisywany w MongoDB musi mieć klucz `"_id"`. Wartość tego klucza może mieć dowolny typ, ale domyślnie otrzymuje typ `ObjectId`. W ramach jednej kolekcji każdy dokument musi mieć unikalną wartość klucza `"_id"`, co sprawia, że każdy dokument w kolekcji może zostać jednoznacznie zidentyfikowany. Oznacza to, że w dwóch kolekcjach mogą znaleźć się dokumenty, których klucz `"_id"` będzie miał wartość `123`. Jednocześnie każda z tych kolekcji będzie mogła mieć tylko jeden dokument, w którym klucz `"_id"` będzie miał wartość `123`.

Typ ObjcetId

`ObjectId` jest domyślnym typem klucza `"_id"`. Klasa `ObjectId` została zaprojektowana tak, żeby pozwalała na łatwe wygenerowanie wartości globalnie unikalnych pomiędzy wieloma komputerami bez nadmiernego obciążania komputera. Rozproszona natura baz danych MongoDB jest powodem porzucenia tradycyjnych rozwiązań typu automatycznie zwiększającego się klucza głównego i stosowania zamiast niego klasy `ObjectId`. Po prostu synchronizowanie automatycznych kluczy głównych między wieloma serwerami jest bardzo trudne i czasochłonne. MongoDB jest systemem od początku projektowanym do obsługi rozproszonych baz danych, dlatego tak ważne było uzyskanie metody generowania jednoznacznych identyfikatorów w tak podzielonym środowisku.

Obiekty typu `ObjectId` zajmują w pamięci 12 bajtów, co pozwala zapisać je jako ciąg znaków składający się z 24 cyfr szesnastkowych (po dwie cyfry na bajt). Sprawia to, że wyglądają one na większe, niż są w rzeczywistości, powodując nerwowość niektórych osób. Trzeba pamiętać o tym, że często pojawiające się wartości `ObjectId` w postaci wielkiego szesnastkowego ciągu znaków tak naprawdę są o połowę mniejsze w pamięci komputera.

Jeżeli szybko wygenerujemy kilka nowych wartości `ObjectId`, to zauważymy, że za każdym razem zmieniają się w nich jedynie ostatnie cyfry. Dodatkowo, jeżeli pomiędzy tworzeniem kolejnych wartości odczekamy kilka sekund, to zauważymy, że zmieniają się też cyfry znajdujące się w środku otrzymywanych liczb. Wynika to z metody generowania wszystkich wartości typu `ObjectId`. Dwanaście bajtów w obiekcie `ObjectId` jest generowanych w następujący sposób:

0	1	2	3	4	5	6	7	8	9	10	11
Znacznik czasu				Losowa liczba					Licznik (zaczyna od wartości losowej)		

Pierwsze cztery bajty w obiekcie to znacznik czasu podany w milisekundach od początku epoki Uniksa. Użycie w tym miejscu znacznika czasu daje nam kilka przydatnych cech:

- Znacznik czasu w połączeniu z następnymi pięcioma bajtami (o których opowiemy za chwilę) pozwala na uzyskanie unikalności z dokładnością do sekundy.
- Dzięki temu, że znacznik czasu znajduje się na początku, wartości `ObjectId` będą sortowane *mniej więcej* w kolejności wstawiania. Nie daje to całkowitej gwarancji, ale ma tę miłą właściwość, że indeksowanie wartości `ObjectId` jest bardzo wydajne.
- Te cztery bajty zawierają też pośrednio informację o czasie utworzenia dokumentu. Większość sterowników udostępnia metodę pobierania tej informacji z wartości `ObjectId`.

Ze względu na to, że w wartościach `ObjectId` zapisywany jest aktualny czas, część użytkowników obawia się o konieczność synchronizowania zegarów w poszczególnych serwerach. Co prawda synchronizowanie zegarów daje nam kilka innych korzyści (zajrzyj do punktu „Synchronizowanie zegarów" w rozdziale 24.), ale dla wartości `ObjectId` sam znacznik czasu nie ma większego znaczenia. Ważne jest tylko to, że często (raz na sekundę) ma on nową wartość, która zawsze się zwiększa.

Następnych pięć bajtów w wartości `ObjectId` to liczba generowana losowo. Ostatnie trzy bajty to licznik, który zaczyna odliczanie od wartości losowej, co ma zapobiegać powstawaniu kolizji między wartościami `ObjectId` generowanymi na różnych komputerach.

Pierwszych dziewięć bajtów w wartościach `ObjectId` gwarantuje zatem unikalność generowanych wartości pomiędzy różnymi komputerami i procesami z dokładnością do jednej sekundy. Ostatnie trzy bajty są już tylko licznikiem odpowiedzialnym za uzyskanie unikalności generowanych wartości w ramach jednej sekundy w jednym procesie. Taka konstrukcja umożliwia wygenerowanie 256^3 (16 777 216) unikalnych wartości `ObjectId` w każdej sekundzie pracy *jednego procesu*.

Automatyczne generowanie wartości dla kluczy _id

Jak już pisaliśmy wcześniej, podczas wstawiania dokumentu do kolekcji nie ma w nim klucza `"_id"`. Taki klucz jest automatycznie dodawany do wstawianego dokumentu. Operacją tą może zajmować się serwer MongoDB, ale zazwyczaj tę pracę wykonuje sterownik działający po stronie klienta.

Używanie powłoki MongoDB

W tym podrozdziale zajmiemy się używaniem powłoki jako części zestawu narzędziowego dla wiersza poleceń. Poznamy sposoby dostosowywania jej do własnych potrzeb i wykorzystywania bardziej zaawansowanych funkcji.

Co prawda powyżej łączyliśmy się z lokalną instancją serwera *mongod*, ale powłoka umożliwia też łączenie się z dowolną inną, osiągalną instancją MongoDB. Aby połączyć się z serwerem *mongod* działającym na innym komputerze albo dostępnym poprzez inny port, podczas uruchamiania powłoki należy podać nazwę hosta, numer portu i nazwę bazy danych:

```
$ mongo inny-host:30000/myDB
MongoDB shell version: 4.2.0
connecting to: inny-host:30000/myDB
>
```

Od teraz zmienna `db` będzie odnosić się do bazy danych *myDB* dostępnej pod adresem *inny-host:3000*.

Czasami przydaje się samo uruchomienie powłoki *mongo*, bez jednoczesnego łączenia się z jakąkolwiek instancją *mongod*. Jeżeli uruchomisz powłokę z parametrem `--nodb`, to nie będzie ona próbowała się łączyć z żadnym serwerem:

```
$ mongo --nodb
MongoDB shell version: 4.2.0
>
```

Po takim uruchomieniu powłoki możesz samodzielnie połączyć się z serwerem *mongod*, wprowadzając polecenie `new Mongo("nazwa-hosta")`:

```
> conn = new Mongo("nazwa-hosta:30000")
connection to nazwa-hosta:30000
> db = conn.getDB("myDB")
myDB
```

Po wprowadzeniu tych dwóch poleceń możesz już normalnie korzystać ze zmiennej `db`. Za pomocą tych poleceń możesz w dowolnym momencie łączyć się z różnymi bazami danych lub serwerami.

Porady przy używaniu powłoki

Dzięki temu, że *mongo* jest powłoką języka JavaScript, wiele pomocy na jej temat można uzyskać, po prostu przeglądając dostępną w sieci dokumentację tego języka. Jeżeli chodzi o funkcje związane z samym MongoDB, to powłoka ma wbudowaną pomoc, z której można skorzystać po wpisaniu polecenia `help`:

```
> help
    db.help()                    help on db methods
    db.mycoll.help()             help on collection methods
    sh.help()                    sharding helpers
...
    show dbs                     show database names
    show collections             show collections in current database
    show users                   show users in current database
...
```

Pomoc dotycząca bazy danych wyświetlana jest przez polecenie `db.help()`, natomiast za pomocą polecenia `db.foo.help()` można uzyskać pomoc dotyczącą kolekcji.

Aby dowiedzieć się, czym właściwie zajmuje się dana funkcja, można po prostu wpisać jej nazwę bez nawiasów. Spowoduje to wypisanie kodu źródłowego tej funkcji. Jeżeli ciekawi Cię na przykład, jak działa funkcja `update`, albo po prostu nie pamiętasz kolejności jej parametrów, to możesz się tego dowiedzieć w ten sposób:

```
> db.movies.updateOne
function (filter, update, options) {
    var opts = Object.extend({}, options || {});

    // Check if first key in update statement contains a $
    var keys = Object.keys(update);
    if (keys.length == 0) {
        throw new Error("the update operation document must contain at
        least one atomic operator");
    }
...
```

Uruchamianie skryptów w powłoce

Oczywiście powłoki można używać interaktywnie, ale dodatkowo pozwala ona na wykonywanie całych skryptów języka JavaScript zapisanych w osobnych plikach. Wystarczy przekazać jej nazwę pliku w wierszu poleceń:

```
$ mongo skrypt1.js skrypt2.js skrypt3.js
MongoDB shell version: 4.2.1
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 4.2.1

loading file: skrypt1.js
Ja jestem skrypt1.js
loading file: skrypt2.js
Ja jestem skrypt2.js
loading file: skrypt3.js
Ja jestem skrypt3.js
...
```

Powłoka *mongo* wykona po kolei wszystkie podane jej skrypty i zakończy swoje działanie.

Jeżeli chcesz uruchomić skrypt, łącząc się jednocześnie z serwerem *mongod* działającym na niestandardowym komputerze lub porcie, to najpierw musisz podać adres serwera, a dopiero potem nazwę skryptu:

```
$ mongo serwer-1:30000/foo --quiet skrypt1.js skrypt2.js skrypt3.js
```

To polecenie spowoduje wykonanie trzech skryptów, podczas gdy zmienna `db` będzie powiązana z bazą danych `foo` dostępną na serwerze *serwer-1:3000*.

W skryptach można wypisywać dane na standardowe wyjście (tak jak robiły to przedstawione wyżej skrypty), używając do tego funkcji `print`. Umożliwia to wykorzystanie powłoki jako elementu w potoku poleceń. Jeżeli chcesz skierować wyjście skryptu powłoki na wejście innego polecenia, to skorzystaj z opcji `--quiet`, aby zablokować domyślne wypisywanie powitania „MongoDB shell version v4.2.0”.

Skrypty można też uruchamiać w powłoce interaktywnej, używając do tego funkcji `load`:

```
> load("skrypt1.js")
Ja jestem skrypt1.js
true
>
```

Skrypty mają dostęp do zmiennej `db` (jak również do wszystkich innych zmiennych globalnych). Pamiętaj jednak, że pomocnicze funkcje powłoki, takie jak `use db` albo `show collections`, nie są dostępne dla plików ze skryptami. W tabeli 2.1 przedstawiam funkcje języka JavaScript równoważne z różnymi pomocniczymi poleceniami powłoki.

Tabela 2.1. Funkcje JavaScript równoważne z pomocniczymi poleceniami powłoki

Polecenie pomocnicze	Równoważna funkcja
`use video`	`db.getSisterDB("video")`
`show dbs`	`db.getMongo().getDBs()`
`show collections`	`db.getCollectionNames()`

Skryptów można też używać do wstrzykiwania zmiennych do powłoki. Na przykład możesz przygotować sobie skrypt, który będzie inicjował często używane przez Ciebie funkcje pomocnicze. Poniższy przykładowy skrypt może okazać się przydatny w trzeciej i czwartej części tej książki. Definiuje on funkcję `connectTo`, która na podanym porcie łączy się z lokalną bazą danych i uzyskane połączenie przypisuje do zmiennej `db`.

```
// defineConnectTo.js
/**
* Połącz się z bazą danych i przypisz połączenie do zmiennej db.
*/
var connectTo = function(port, dbname) {
    if (!port) {
        port = 27017;
    }

    if (!dbname) {
        dbname = "test";
    }

    db = connect("localhost:"+port+"/"+dbname);
    return db;
};
```

Po załadowaniu tego skryptu do powłoki pojawi się definicja zmiennej `connectTo`:

```
> typeof connectTo
undefined
> load('defineConnectTo.js')
> typeof connectTo
function
```

Skrypty można wykorzystać do dodawania nowych funkcji pomocniczych, ale też do automatyzowania często wykonywanych zadań i działań administracyjnych.

Domyślnie powłoka będzie przeglądać katalog, w którym została uruchomiona (za pomocą funkcji `pwd()` możesz odczytać nazwę tego katalogu). Jeżeli Twój skrypt nie znajduje się w aktualnym katalogu, to możesz podać powłoce względną lub bezwzględną ścieżkę do tego pliku. Na przykład, jeżeli umieścimy swoje skrypty powłoki w katalogu *~/moje-skrypty*, to plik *defineConnectTo.js* będzie można załadować za pomocą polecenia `load("/home/katalog-użytkownika/moje-skrypty/defineConnectTo.js")`. Pamiętaj, że funkcja `load` nie może korzystać ze znaku tyldy (~).

Można też używać funkcji `run`, aby w ten sposób uruchamiać w powłoce programy wiersza poleceń. Argumenty przekazane funkcji staną się parametrami dla wywoływanego programu:

```
> run("ls", "-l", "/home/katalog-użytkownika/moje-skrypty/")
sh70352| -rw-r--r-- 1 myUser myUser 2012-12-13 13:15 defineConnectTo.js
sh70532| -rw-r--r-- 1 myUser myUser 2013-02-22 15:10 skrypt1.js
sh70532| -rw-r--r-- 1 myUser myUser 2013-02-22 15:12 skrypt2.js
sh70532| -rw-r--r-- 1 myUser myUser 2013-02-22 15:13 skrypt3.js
```

Ta funkcja ma niestety ograniczone zastosowanie, ponieważ jej wyjście jest dziwnie sformatowane i nie pozwala ona na stosowanie potoków.

Tworzenie pliku .mongorc.js

Jeżeli masz pewne często ładowane skrypty, to możesz umieścić je w pliku *.mongorc.js*, który jest wykonywany przy każdym uruchomieniu powłoki.

Załóżmy, że chcesz, żeby powłoka odpowiednio witała Cię za każdym razem, gdy się do niej logujesz. Utwórz zatem w swoim katalogu użytkownika plik o nazwie *.mongorc.js* i wprowadź do niego poniższy kod:

```
// .mongorc.js

var compliment = ["atrakcyjnie", "inteligentnie", "jak Batman"];
var index = Math.floor(Math.random()*3);

print("Witaj, dzisiaj wyglądasz wyjątkowo "+compliment[index]+"!");
```

Po uruchomieniu powłoki zobaczysz coś takiego:

```
$ mongo
MongoDB shell version: 4.2.1
connecting to: test
Witaj, dzisiaj wyglądasz wyjątkowo jak Batman!
>
```

Bardziej praktyczne będzie wykorzystanie tego skryptu do konfigurowania często używanych zmiennych globalnych, przygotowywania krótkich aliasów dla długich nazw albo podmieniania wbudowanych funkcji. Jednym z typowych zastosowań pliku *.mongorc.js* jest usuwanie z powłoki co bardziej „niebezpiecznych" poleceń pomocniczych. Można w ten sposób unieszkodliwić takie polecenia jak `dropDatabase` albo `deleteIndexes`, podmieniając je pustymi funkcjami albo całkowicie usuwając te definicje:

```
var no = function() {
    print("Nie na mojej zmianie!");
};
```

```
// Zapobiega usuwaniu baz danych
db.dropDatabase = DB.prototype.dropDatabase = no;

// Zapobiega usuwaniu kolekcji
DBCollection.prototype.drop = no;

// Zapobiega usuwaniu indeksu
DBCollection.prototype.dropIndex = no;

// Zapobiega usuwaniu indeksów
DBCollection.prototype.dropIndexes = no;
```

Jeżeli teraz spróbujesz wywołać dowolną z tych funkcji, to po prostu pojawi się komunikat o błędzie. Pamiętaj, że ta technika nie uchroni Cię przez złośliwymi użytkownikami. Jest to jedynie proste zabezpieczenie przed nieprzemyślanymi działaniami.

Możesz też pominąć ładowanie pliku *.mongorc.js*, podając opcję `--norc` podczas uruchamiania powłoki.

Dostosowywanie znaku zachęty

Domyślny wygląd znaku zachęty w powłoce może zostać zmieniony przez przypisanie zmiennej `prompt` innego ciągu znaków lub funkcji. Na przykład, jeżeli musisz uruchamiać zapytanie, którego wykonanie zajmuje całe minuty, to możesz sobie przygotować znak zachęty zawierający aktualny czas, aby móc zobaczyć, kiedy zakończyło pracę poprzednie polecenie:

```
prompt = function() {
    return (new Date())+"> ";
};
```

Inna przydatna forma znaku zachęty może wypisywać nazwę aktualnie używanej bazy danych:

```
prompt = function() {
    if (typeof db == 'undefined') {
        return '(nodb)> ';
    }

    // Sprawdź wynik ostatniej operacji na bazie danych
    try {
        db.runCommand({getLastError:1});
    }
    catch (e) {
        print(e);
    }

    return db+"> ";
};
```

Pamiętaj, że funkcje tworzące znak zachęty powinny zwracać ciąg znaków i bardzo uważać na ewentualne wyjątki. Dla użytkownika pojawienie się wyjątku w miejscu znaku zachęty z pewnością będzie bardzo dziwnym doświadczeniem.

W funkcji tworzącej znak zachęty warto też odwoływać się do funkcji `getLastError`. Pozwala to na wykrycie błędów przy zapisie i powoduje automatyczne ponowne połączenie w przypadku, gdy powłoka zostanie rozłączona (na przykład z powodu ponownego uruchomienia serwera *mongod*).

Jeżeli zawsze chcesz używać zmienionego znaku zachęty, to odpowiednią modyfikację dobrze jest umieścić w pliku *.mongorc.js*. Możesz też przygotować sobie kilka różnych definicja znaku zachęty i wymieniać je w powłoce w razie potrzeby.

Edytowanie złożonych zmiennych

Dostępna w powłoce obsługa poleceń wielowierszowych jest trochę ograniczona. Nie pozwala ona na edytowanie wprowadzonych wcześniej wierszy, co może być problemem, jeżeli wprowadzając piętnasty wiersz, zauważysz w pierwszym drobną pomyłkę. Z tego powodu większe bloki kodu lub całe obiekty dobrze jest wprowadzać w osobnym edytorze. W tym celu zdefiniuj w powłoce zmienną `EDITOR` (możesz też to zrobić w swoim środowisku, ale skoro już pracujemy w powłoce…):

```
> EDITOR="/usr/bin/emacs"
```

Od teraz, chcąc edytować dowolną zmienną, możesz użyć polecenia `edit` *`nazwa_zmiennej`*. Na przykład tak:

```
> var wap = db.books.findOne({title: "Wojna i pokój"});
> edit wap
```

Po zakończeniu wprowadzania zmian wystarczy je zapisać i zamknąć edytor. Uzyskany tekst zostanie sparsowany i załadowany do wskazanej zmiennej.

Dopisz jeszcze polecenie `EDITOR="`*`/ścieżka/do/edytora`*`"` w pliku *.mongorc.js*, aby nie było więcej konieczności samodzielnego konfigurowania edytora zmiennych.

Niewygodne nazwy kolekcji

Pobieranie kolekcji za pomocą składni `db.`*`nazwaKolekcji`* działa niemal zawsze, chyba że kolekcja nosi nazwę będącą zarezerwowanym słowem kluczowym albo niepoprawną nazwą właściwości języka JavaScript.

Załóżmy, że chcemy uzyskać dostęp do kolekcji o nazwie `version`. W takiej sytuacji nie możemy użyć polecenia `db.version`, ponieważ `db.version` jest metodą zmiennej `db`, która zwraca numer wersji działającego serwera MongoDB:

```
> db.version
function () {
    return this.serverBuildInfo().version;
}
```

Aby rzeczywiście pobrać kolekcję o nazwie `version`, musimy skorzystać z funkcji `getCollection`:

```
> db.getCollection("version");
test.version
```

Tej metody można używać również do pobierania kolekcji, których nazwy nie są poprawnymi nazwami właściwości w języku JavaScript, takimi jak *foo-bar-baz* albo *123abc*. (Nazwy właściwości w języku JavaScript mogą zawierać wyłącznie litery, cyfry i znaki dolara lub podkreślenia, a dodatkowo nie mogą zaczynać się od cyfry).

Inną metodą radzenia sobie z niepoprawnymi nazwami właściwości jest wykorzystanie składni dostępu do tablic. W języki JavaScript zapis `x.y` jest równoznaczny z zapisem `x['y']`. Oznacza to, że dostęp do podkolekcji można uzyskać za pomocą zmiennych, a nie wyłącznie za pomocą literałów. A zatem w przypadku, gdy pewną operację musimy wykonać na każdej podkolekcji z grupy `blog`, możemy iterować po nich za pomocą poniższego kodu:

```
var collections = ["posts", "comments", "authors"];

for (var i in collections) {
    print(db.blog[collections[i]]);
}
```

Ten zapis można wykorzystać zamiast jawnego zapisu, takiego jak poniżej:

```
print(db.blog.posts);
print(db.blog.comments);
print(db.blog.authors);
```

Zauważ, że nie można w tym miejscu użyć zapisu `db.blog.i`, ponieważ zostałby on zinterpretowany jako nazwa kolekcji `test.blog.i`, a nie `test.blog.posts`. Jeżeli litera `i` ma być traktowana jak zmienna, to trzeba skorzystać ze składni `db.blog[i]`.

Tej techniki można używać również w przypadku kolekcji o bardzo dziwacznych nazwach:

```
> var name = "@#&!"
> db[name].find()
```

Próba uzyskania dostępu do kolekcji za pomocą polecenia `db.@#&!` skończyłaby się błędem, ale zapis `db[name]` działa znakomicie.

MongoDB – przekonaj się na własnym systemie!

MongoDB jest wieloplatformowym, nierelacyjnym systemem do obsługi baz danych, napisanym w języku C++. Nie przypomina ściśle ustrukturyzowanych relacyjnych baz danych, zamiast tego korzysta z dokumentów w formacie BSON. Ułatwia to bardziej naturalne przetwarzanie informacji w aplikacjach, oczywiście przy zachowaniu możliwości tworzenia hierarchii oraz indeksowania. W ten sposób cały system zyskuje na wydajności, co jest szczególnie istotne przy przetwarzaniu bardzo dużych zbiorów danych. MongoDB umożliwia stosowanie elastycznych modeli danych, uzyskiwanie wysokiego poziomu dostępności i poziome skalowanie.

Ten praktyczny przewodnik jest przeznaczony dla użytkowników bazy MongoDB w wersji 4.2. W przystępny i konkretny sposób opisuje zalety stosowania dokumentowych baz danych, równocześnie wskazuje zaawansowane metody konfiguracji systemu oraz możliwe zastosowania w różnych projektach. Książka zainteresuje zarówno użytkowników i administratorów MongoDB, jak i programistów tworzących złożone aplikacje. Przedstawia kwestie tworzenia zapytań, indeksów, agregacji, transakcji, zbiorów replik, zarządzania systemem, shardingu i administrowania danymi, trwałości danych, monitorowania systemu oraz jego zabezpieczenia. Znalazło się tu także wprowadzenie do pracy z MongoDB, omówiono też zasady pracy z klastrem shardów oraz administrowania aplikacją i serwerem bazy MongoDB.

W książce między innymi:

- ogólne zasady pracy z MongoDB
- operacje zapisu i wyszukiwania oraz tworzenie złożonych zapytań
- indeksy w kolekcjach, agregowanie danych i transakcje
- lokalny zbiór replik i korzystanie z replikacji
- konfiguracja elementów klastra
- monitorowanie systemu, kopie bezpieczeństwa i odtwarzanie bazy MongoDB

Shannon Bradshaw jest wiceprezesem firmy MongoDB. Zarządza szkoleniami prowadzonymi przez MongoDB University w ramach programu MongoDB Professional Certification. Wcześniej był wykładowcą na wyższych uczelniach.

Eoin Brazil zajmuje się dydaktyką w firmie MongoDB. Pracuje nad szkoleniami sieciowymi oraz bezpośrednimi.

Kristina Chodorow jest inżynierem oprogramowania w firmie Google. Rozwijała zbiory replik MongoDB i napisała sterowniki dla języków PHP i Perl. Wypowiadała się na temat MongoDB na międzynarodowych konferencjach.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-6533-9

9 788328 365339

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 89,00 zł