# Podstawowa administracja bazą danych - mongosh

**Wprowadzenie:**

mongosh to interaktywna powłoka MongoDB oparta na JavaScripcie, służąca do administrowania serwerem i pracy z danymi. Zastąpiła starą powłokę `mongo`. Poniżej zebrano podstawowe komendy administracyjne — każda osobno, z opisem działania. Instrukcja zakłada pracę z hosta; jeśli serwer działa w kontenerze Docker, zobacz sekcję 1, która wyjaśnia jak „przetłumaczyć" każdą komendę uruchomieniową.

# 1. Dwa konteksty: baremetal vs kontener Docker

Sposób *wejścia* do powłoki zależy od tego, gdzie zainstalowany jest mongosh. Reszta komend (te zaczynające się od `db.` oraz `show ...`) wykonywana jest już wewnątrz powłoki i wygląda tak samo w obu przypadkach.

**Baremetal** — mongosh zainstalowany bezpośrednio na hoście:

```
mongosh
```

**Docker** — mongosh znajduje się w kontenerze (np. o nazwie `mongo`), więc komendę poprzedzamy wywołaniem `docker exec`:

```
docker exec -it mongo mongosh
```

**Zasada przekładu:** dowolną komendę `mongosh ...` z tej instrukcji można uruchomić w kontenerze, poprzedzając ją `docker exec -it <nazwa_kontenera>` (tryb interaktywny) lub samym `docker exec <nazwa_kontenera>` w przypadku jednolinijkowców `--eval`.

# 2. Komenda `mongosh` – uruchomienie powłoki

```
mongosh
```

Uruchamia interaktywną powłokę i łączy się z domyślnym serwerem (`localhost:27017`). Po połączeniu pojawia się znak zachęty, w który wpisuje się kolejne komendy.

**Z uwierzytelnieniem** (powłoka zapyta o hasło):

```
mongosh -u admin -p --authenticationDatabase admin
```

Opcja `-u` podaje użytkownika, `-p` bez wartości wymusza interaktywny monit o hasło (bezpieczniej niż wpisywanie go w linii poleceń), a `--authenticationDatabase` wskazuje bazę, w której zdefiniowano konto.

**Pojedyncza komenda bez wchodzenia do powłoki** (przydatne w skryptach):

```
mongosh --quiet --eval 'db.version()'
```

Opcja `--eval` wykonuje przekazany kod i kończy działanie; `--quiet` wycisza baner powitalny. W kontenerze: `docker exec mongo mongosh --quiet --eval 'db.version()'`.

> **Screen:** ekran po udanym połączeniu z powłoką.

# 3. Komenda `show dbs` – lista baz danych

```
show dbs
```

Wyświetla listę wszystkich baz danych na serwerze wraz z ich rozmiarem na dysku. Bazy puste (bez zapisanych danych) mogą się nie pojawić na liście.

**Przykład:**

```
test> show dbs
admin   180.00 KiB
config   60.00 KiB
local    72.00 KiB
```

# 4. Komenda `use` – wybór bazy danych

```
use admin
```

Przełącza kontekst na wskazaną bazę. Od tego momentu zmienna `db` odnosi się do tej bazy. Jeśli baza nie istnieje, zostanie utworzona dopiero przy pierwszym zapisie danych.

# 5. Komenda `db` – bieżąca baza

```
db
```

Wyświetla nazwę bazy danych, na której aktualnie pracujemy. Pomaga zorientować się w kontekście przed wykonaniem operacji modyfikujących dane.

# 6. Komenda `show collections` – lista kolekcji

```
show collections
```

Wyświetla wszystkie kolekcje (odpowiednik tabel) w bieżącej bazie. Wymaga wcześniejszego wybrania bazy poleceniem `use`.

# 7. Komenda `db.getUsers()` – listowanie użytkowników

```
use admin
db.getUsers()
```

Wyświetla użytkowników zdefiniowanych w bieżącej bazie wraz z ich rolami. Konta administracyjne zwykle żyją w bazie `admin`, dlatego najpierw przełączamy się do niej poleceniem `use admin`.

**Skrót shellowy** (to samo działanie):

```
show users
```

**Pełniejsze informacje o wszystkich kontach:**

```
db.runCommand({ usersInfo: 1 })
```

Zwraca szczegółowy wykaz użytkowników bieżącej bazy w formie surowego dokumentu.

> **Screen:** wynik `db.getUsers()` na bazie admin.

# 8. Komenda `db.createUser()` – tworzenie użytkownika

```
db.createUser({
  user: "backup",
  pwd: passwordPrompt(),
  roles: [ { role: "backup", db: "admin" } ]
})
```

Tworzy nowe konto w bieżącej bazie. Funkcja `passwordPrompt()` powoduje, że powłoka zapyta o hasło interaktywnie, zamiast zapisywać je w historii poleceń. Pole `roles` to lista par rola–baza określająca uprawnienia konta.

# 9. Komenda `db.dropUser()` – usuwanie użytkownika

```
db.dropUser("backup")
```

Usuwa wskazane konto z bieżącej bazy. Operacja jest nieodwracalna, dlatego należy upewnić się, że pracujemy na właściwej bazie (sprawdź poleceniem `db`).

# 10. Komendy `db.grantRolesToUser()` / `db.revokeRolesFromUser()` – zarządzanie rolami

```
db.grantRolesToUser("backup", [ { role: "readWrite", db: "appdb" } ])
```

Nadaje istniejącemu użytkownikowi dodatkową rolę na wskazanej bazie — tutaj prawo odczytu i zapisu w bazie `appdb`.

```
db.revokeRolesFromUser("backup", [ { role: "readWrite", db: "appdb" } ])
```

Odbiera wcześniej nadaną rolę. Składnia jest identyczna jak przy nadawaniu.

# 11. Komenda `db.changeUserPassword()` – zmiana hasła

```
db.changeUserPassword("backup", passwordPrompt())
```

Zmienia hasło istniejącego użytkownika. Ponownie warto użyć `passwordPrompt()`, aby nie ujawniać hasła w linii poleceń ani w historii.

# 12. Komenda `db.getRoles()` – przegląd ról

```
db.getRoles()
```

Wyświetla role zdefiniowane w bieżącej bazie. Oprócz ról własnych MongoDB udostępnia role wbudowane, których ta komenda standardowo nie pokazuje.

> **Typowe role wbudowane:** `read`, `readWrite`, `dbAdmin`, `userAdmin`, `dbOwner` (na konkretnej bazie) oraz globalne `readWriteAnyDatabase`, `userAdminAnyDatabase`, `backup`, `restore`, `root`.

# 13. Komenda `db.runCommand({ connectionStatus: 1 })` – kim jestem

```
db.runCommand({ connectionStatus: 1 })
```

Zwraca informacje o bieżącym połączeniu: zalogowanego użytkownika oraz przyznane mu role. Przydatne do szybkiej weryfikacji, czy uwierzytelnienie powiodło się i z jakimi uprawnieniami pracujemy.

# 14. Komenda `db.stats()` – statystyki bazy

```
db.stats()
```

Wyświetla statystyki bieżącej bazy: liczbę kolekcji, liczbę dokumentów oraz zajętość miejsca na dysku. Pomocne przy ocenie rozmiaru bazy przed wykonaniem kopii zapasowej.

> **Screen:** wynik `db.stats()`.

# 15. Komenda `db.serverStatus()` – status serwera

```
db.serverStatus()
```

Zwraca obszerny dokument z metrykami całego serwera: połączenia, zużycie pamięci, operacje, stan replikacji. Wynik jest bardzo duży, dlatego zwykle odczytuje się z niego pojedyncze pola, np. `db.serverStatus().connections`.

# 16. Komenda `db.<kolekcja>.find()` – wyszukiwanie dokumentów

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

Wyświetla dokumenty z kolekcji. Bez argumentów zwraca wszystkie; w mongosh wyniki są domyślnie czytelnie sformatowane.

**Z filtrem i operatorami porównań:**

```
db.uzytkownicy.find({ wiek: { $gt: 25 } })          // wiek > 25
db.uzytkownicy.find({ wiek: { $gte: 18, $lt: 65 } }) // zakres
db.uzytkownicy.find({ imie: { $in: ["Anna", "Jan"] } })
```

Operatory `$gt`, `$gte`, `$lt`, `$lte`, `$ne`, `$in` pozwalają budować warunki bardziej złożone niż proste dopasowanie wartości.

> **Screen:** przykładowy wynik `find()`.

# 17. Komendy `db.<kolekcja>.insertOne()` / `insertMany()`

# – wstawianie

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

Wstawia pojedynczy dokument do kolekcji. Jeśli kolekcja nie istnieje, zostanie utworzona automatycznie.

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

Wstawia wiele dokumentów jednocześnie, przekazanych jako tablica.

# 18. Komendy

## `db.<kolekcja>.updateOne()` / `updateMany()` – aktualizacja

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

Aktualizuje pierwszy dokument pasujący do filtra. Operator `$set` ustawia wartość pola; inne przydatne operatory to `$inc` (zwiększ), `$unset` (usuń pole), `$push` i `$pull` (operacje na tablicach).

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

Aktualizuje wszystkie dokumenty pasujące do filtra. Dodanie opcji `{ upsert: true }` jako trzeciego argumentu sprawia, że dokument zostanie utworzony, jeśli żaden nie pasuje.

# 19. Komendy

## `db.<kolekcja>.deleteOne()` / `deleteMany()` – usuwanie

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

Usuwa pierwszy dokument pasujący do filtra.

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

Usuwa wszystkie dokumenty pasujące do filtra. Pusty filtr `{}` usunąłby całą zawartość kolekcji, dlatego należy zachować ostrożność.

# 20. Komenda

## `db.<kolekcja>.createIndex()` – indeksy

```
db.uzytkownicy.createIndex({ imie: 1 })
```

Tworzy indeks na wskazanym polu (`1` = rosnąco, `-1` = malejąco), co przyspiesza wyszukiwanie po tym polu.

```
db.uzytkownicy.createIndex({ email: 1 }, { unique: true })
```

Tworzy indeks z wymuszeniem unikalności wartości w danym polu.

```
db.uzytkownicy.getIndexes()
db.uzytkownicy.dropIndex("imie_1")
```

Pierwsza komenda wypisuje istniejące indeksy kolekcji, druga usuwa indeks po jego nazwie.

# Podsumowanie praktyczne

```
# Wejście do powłoki (baremetal)
mongosh -u admin -p --authenticationDatabase admin

# ...lub w kontenerze
docker exec -it mongo mongosh -u admin -p --authenticationDatabase admin
```

```
// Wewnątrz powłoki: rozeznanie i zarządzanie userami
show dbs
use admin
db.getUsers()
db.createUser({ user: "backup", pwd: passwordPrompt(), roles: [ { role: "backup", db: "admin" } ] })
db.runCommand({ connectionStatus: 1 })
```

---

Revision #1
Created 9 June 2026 09:14:54 by Tomasz Konieczny
Updated 9 June 2026 09:24:31 by Tomasz Konieczny