UNIVERZITET U ZENICI
Politehnički fakultet
Softversko inženjerstvo
Optimizacija baza podataka

Optimizacija i analiza performansi Microsoft SQL Server upita u kontekstu PyQT aplikacije

Projektni zadatak

Student: **Edina Kaknjo**

Profesor: Doc. dr. sc. Denis Čeke

Zenica, akademska 2024/2025. godina

SADRŽAJ

1. UVOD

U savremenom razvoju softverskih sistema i rješenja, optimizacija rada sa bazama podataka predstavlja ključan faktor u postizanju efikasnosti rada naših aplikacija. Microsoft SQL Server se ponekad koristi kao centralni sistem za upravljanje podacima, a performanse njegovih upita direktno utiču na brzinu rada aplikacije. U ovom radu će se koristiti isti.

U okviru ovog rada, razmatra se analiza različitih tehnika za poboljšanje performansi, pri čemu se polazi od pretpostavki da pojedini pristupi mogu dati mjerljive prednosti u odnosu na druge. Kroz rad ću postaviti tri hipoteze, te iste pokušati dokazati.

Rad se temelji na ispitivanju odnosa između načina strukturiranja i izvršavanja upita i njihovog efekta na vrijeme odziva sistema. Na taj način stvara se osnova za donošenje zaključaka o najefikasnijim metodama u različitim scenarijima.

Posebna pažnja treba i biće posvećena CRUD (Create, Read, Update, Delete) operacijama, koje čine osnovu svakog sistema za upravljanje bazama podataka.

Za pokazne potrebe performansi različitih pristupa čitanju i radom sa podataka, kreirat ću jednostavnu PyQT aplikaciju i povezati je direktno sa serverom i bazom.

2. PREGLED BAZE PODATAKA

Za potrebe ovog projekta korištena je Stack Overflow baza podataka (2013 Mini verzija), preuzeta iz službenog sajta i pripremljena za korištenje. Ova baza jepogodna za korištenje na svim verzijama SQL Servera od 2008. godine pa nadalje. Baza je dostupna u obliku .mdf i .ldf fajlova, čime je izbjegnuta potreba za restauracijom iz .bak kopije, te je omogućeno jednostavno pokretanje i testiranje

Karakteristike baze:

- Sadrži podatke samo sa **StackOverflow.com** sajta, bez podataka sa ostalih Stack Exchange stranica.

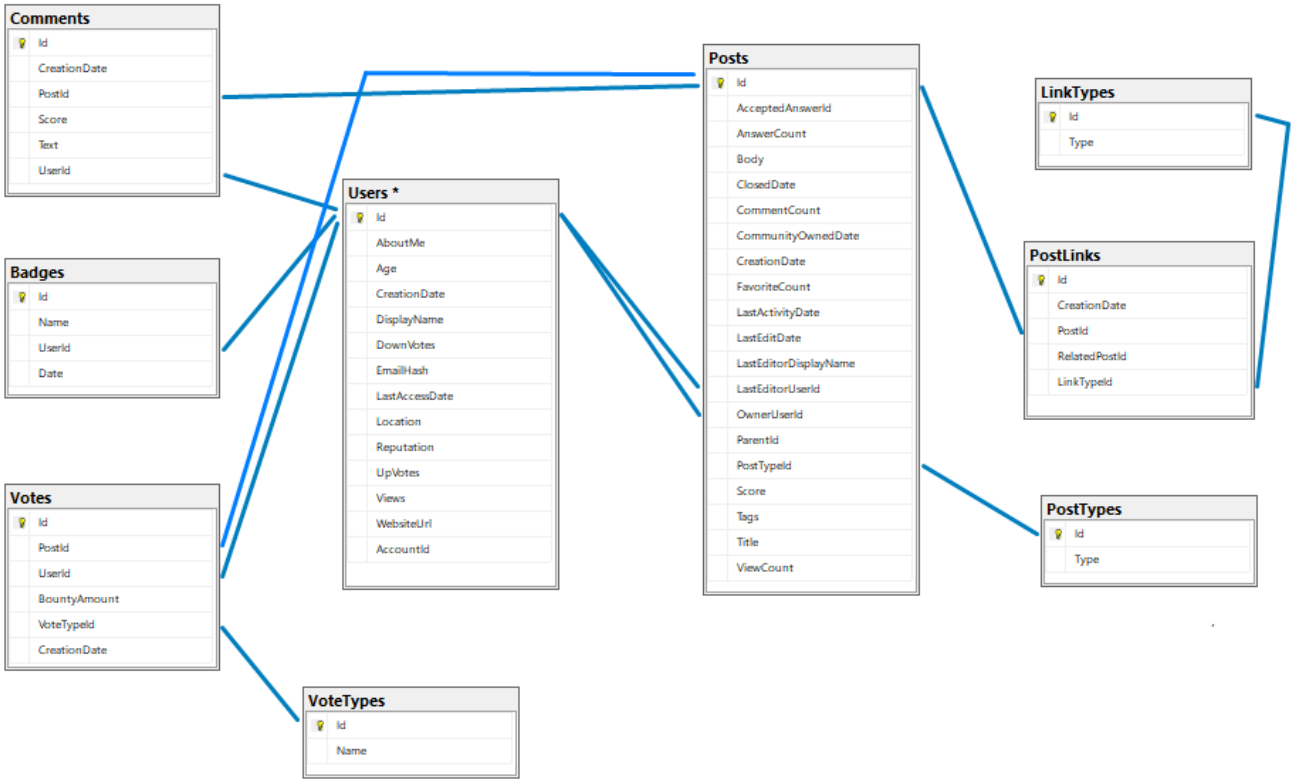
- Svi primarni entiteti (tabele) imaju kreirane clustered indexe, dok nonclustered i full-text indeksi nisu uključeni, čime se smanjila veličina fajlova i olakšalo inicijalno korištenje.

- Log fajl (transaction log) je inicijalno vrlo mali, ali se može proširiti ukoliko je planirano intenzivnije mijenjanje podataka.

Ova verzija baze predstavlja umanjenu, ali funkcionalnu repliku stvarne Stack Overflow baze, sačuvanu radi istraživanja, edukacije i testiranja SQL upita.

Strani ključevi u samoj bazi, djeluje kao da nisu povezani, s obzirom na to bilo je nemoguće automatski generisati dijagram u Microsoft SQL serveru. Na slici 1 su prikazane tabele i njihovi atributi, a linije koje povezuju tabele ne prikazuju nužno odnose (parent- child ili nešto slično), već je moja vizuelna prezentacija ‘poveznica’ među tabelama. Dakle, slika ispod ne prikazuje detalje samog odnosa, već da neki odnos postoji.

Dakle, baza sadrži 9 tabela: Comments, Badges, Votes, VoteTypes, Users, Posts, LinkTypes, PostLinks i PostTypes. Atributi svake od ovih tabela su također prikazani na slici 1.



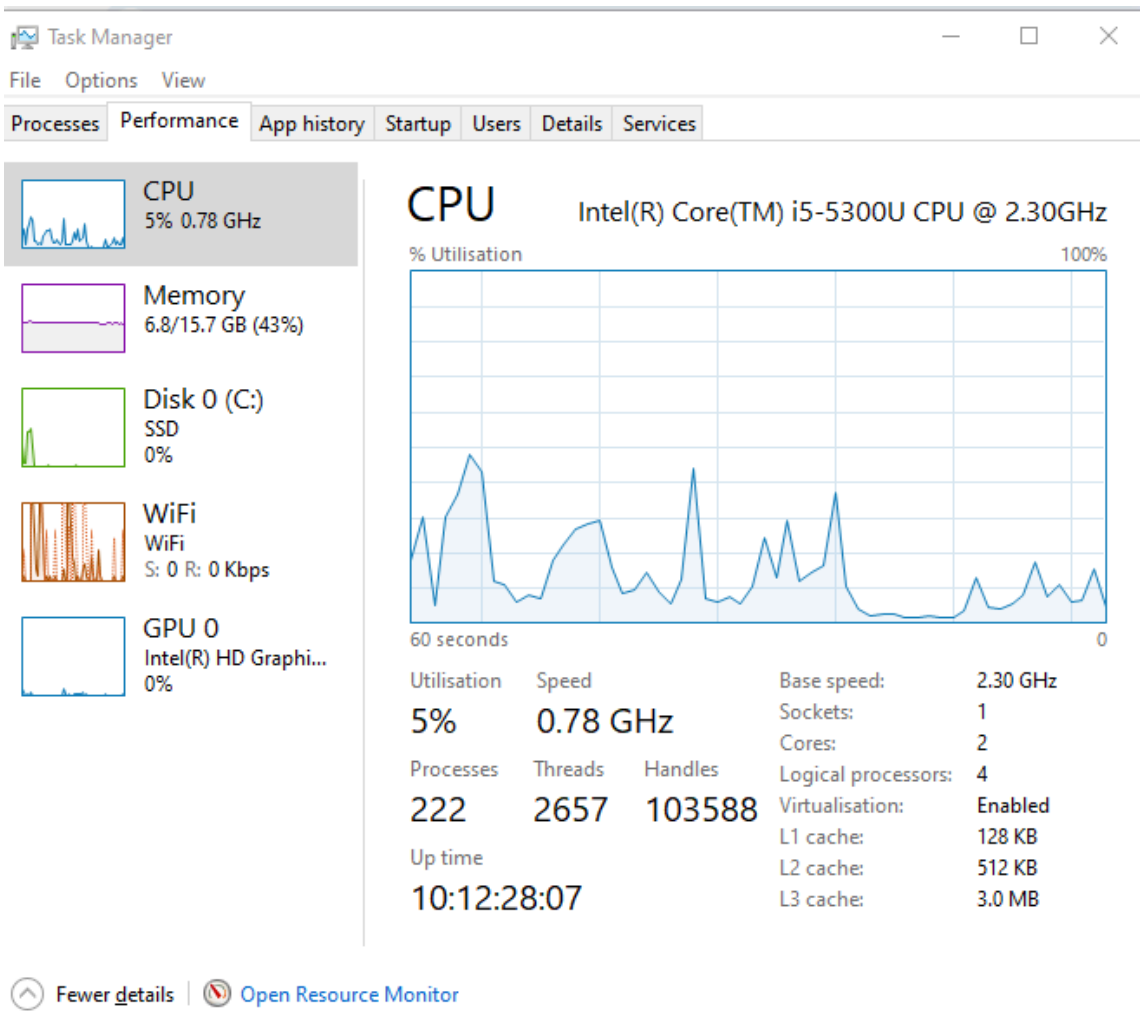
Slika 1. Tabele Stack Overflow baze i odnosi među njima

3. PERFORMANSE MOG UREĐAJA

Testiranja u ovom radu vršena su na ličnom laptopu čije se osnovne karakteristike mogu vidjeti na slici (Task Manager- Performance tab).

Procesor koji pokreće uređaj je Intel Core i5-5300U CPU @ 2.30 GHz sa dvije fizičke jezgre i četiri logička procesora (Hyper-Threading), te sa 3 MB L3 cache memorije. Uređaj raspolaže sa 16 GB RAM memorije, od čega je tokom testiranja prosječna zauzetost bila između 40% i 50%. Kao primarni disk koristi se SSD, što značajno utiče na brzinu čitanja i upisa podataka u poređenju sa klasičnim HDD diskom. Za grafički prikaz zadužena je integrisana kartica Intel HD Graphics, koja je sasvim dovoljna za ovu vrstu testiranja jer se fokus rada nalazi na procesorskim i memorijskim operacijama.

Upravo zbog ovih karakteristika uređaja, mjerene performanse upita se mogu smatrati reprezentativnim za prosječne poslovne laptope srednje klase, iako bi na jačim procesorima i sa boljim optimizacijama rezultati vjerovatno bili kraći i stabilniji.



Slika 2. Karakteristike mog uređaja

4. HIPOTEZA 1

Stored Procedure imaju bolje performanse u odnosu na ad hoc SQL pozive iz aplikacije. U slučaju čestih agregacijskih upita nad istim podacima (npr. brojanje postova po korisniku), bolje performanse se postižu ako se upit definiše kao Stored Procedure u SQL Serveru. Stored Procedure omogućavaju keširanje plana izvršavanja i smanjuju potrebu za parsiranjem i kompilacijom SQL naredbe svaki put. Ova metoda doprinosi bržem odgovoru sistema u realnim scenarijima korištenja.

Ovo je moja prva hipoteza. U trenutku testiranja rezultata na pojedinim upitima aplikacija još uvijek nije kreirana. S obzirom na to hipoteza se može testirati već unutar SQL Server Management Studio (SSMS) ili nekog query alata, bez aplikacije, jer u biti- isti je princip. Samom izvršavanju upita u SSMS je onda samo potrebno dodati vrijeme potrebno za slanje zahtjeva iz aplikacije u SSMS (ili nešto drugo- ovisno u načinu rada finalne aplikacije). Samim time, ukoliko se dokaže da za stored procedure treba manje vremena neko za ad hoc pozive (obične upite), naravno da slijedi i da za stored procedure treba manje vremena u odnosu na ad hoc SQL pozive iz aplikacije, tj da stored procedure imaju bolje performanse.
Također, za testiranje je upotrebljen i SQLQueryStress, opensource C# aplikacija. Ova aplikacija je poslužila kao inicijalna ideja za kreiranje moja aplikacije, samo korištenjem Python programskog jezika, te će se kasnija testiranja vršiti u mojoj aplikaciji.

4.1. Osnovni pojmovi

- *Ad hoc SQL upiti*- SQL naredbe koje aplikacija ili korisnik direktno šalju SQL Serveru, bez prethodnog definisanja u bazi. Kod svakog izvršavanja, server mora da parsira upit, provjeri sintaksu, generiše plan izvršavanja i potom ga izvrši.
  Može dovesti do ponavljanog opterećenja sistema prilikom učestalih i identičnih upita.
- *Stored Procedure*- unaprijed definisane SQL skripte koje se trajno čuvaju u bazi podataka. One omogućavaju ponovnu upotrebu koda bez ponovnog pisanja, keširanje plana izvršavanja (execution plan caching), lakše održavanje.
- *Execution Plan*- način na koji SQL Server odlučuje kako će izvršiti upit. Uključuje informacije o redoslijedu čitanja podataka, korištenim indeksima, troškovima (cost) operacija.

4.2. Način mjerenja performansi

Za upoređivanje performansi koristi se:

- Client Statistics - pokazuje CPU vrijeme i ukupno vrijeme izvršavanja
- Live Query Statistics - daje informacije o čitanju stranica iz memorije ili diska

- SQLQueryStress - trajanje svake egzekucije pod opterećenjem, više korisnika odjednom, itd.. Moguće preuzimanje na: https://github.com/ErikEJ/SqlQueryStress/releases
- Vlastita aplikacija koja je klon SQLQueryStress ali ima dodatne funkcionalnosti i urađena je drugim programskim jezikom.

4.3. Testiranje hipoteze

4.3.1. Test H1_Q1_SP1

H1 Označava hipotezu 1, Q1 označava upit 1, dok SP1 označava stored procedure 1.

Objave (Posts) su nasoj bazi podataka su nastale od 2008-07-31 21:42:52.667 do 2013-12-31 23:59:48.203. Naredni AD HOC upit je pokrenut direktno u SSMS.

```
DECLARE @FromDate datetime2 = '2012-09-04';
DECLARE @ToDate   datetime2 = '2013-09-04';
DECLARE @MinScore int = 5;          -- minimalni score
DECLARE @PostType int = NULL;       -- svi tipovi

-- AD HOC: broj postova po korisniku, sa filtrima
SELECT p.OwnerUserId,
       COUNT(*) AS PostCount
FROM dbo.Posts AS p
WHERE (@FromDate IS NULL OR p.CreationDate >= @FromDate)
  AND (@ToDate   IS NULL OR p.CreationDate <  @ToDate)
  AND (@MinScore IS NULL OR p.Score >= @MinScore)
  AND (@PostType IS NULL OR p.PostTypeId = @PostType)
GROUP BY p.OwnerUserId
ORDER BY PostCount DESC;
```

Naredna stored procedura je pokrenuta više puta i korištena sa istim filterima kao prethodni AD HOC upit. Potrebno vrijeme za kreiranje same Stored procedure je ispod 1s.

```
CREATE OR ALTER PROCEDURE dbo.usp_PostCounts
  @FromDate  datetime2 = NULL,
  @ToDate    datetime2 = NULL,
  @MinScore  int       = NULL,
  @PostType  int       = NULL
AS
BEGIN
  SET NOCOUNT ON;

  SELECT p.OwnerUserId,
         COUNT(*) AS PostCount
  FROM dbo.Posts AS p
  WHERE (@FromDate IS NULL OR p.CreationDate >= @FromDate)
    AND (@ToDate   IS NULL OR p.CreationDate <  @ToDate)
    AND (@MinScore IS NULL OR p.Score >= @MinScore)
    AND (@PostType IS NULL OR p.PostTypeId = @PostType)
  GROUP BY p.OwnerUserId
  ORDER BY PostCount DESC;
END
GO
```

4.3.2. Test H1_Q2_SP2

H1 Označava hipotezu 1, Q2 označava upit 2, dok SP2 označava stored procedure 2.

Naredni AD HOC upit dodaje novi post u tabelu Posts,uzimamo OwnerUserId korisnika na osnovu emaila iz Users tabele. Time kombinujemo INSERT + SELECT u jednom upitu. Nailazi se na problem sa kolonom Id u tabeli Posts (SQL Server automatski generiše vrijednosti). Ne može se ručno raditi INSERT INTO Posts (Id, ...), pa se dopušta da SQL sam generiše ID.

```
INSERT INTO Posts (PostTypeId, OwnerUserId, CreationDate, Score, ViewCount, Title, Body)
SELECT
    1,                              -- tip posta: pitanje
    U.Id,                           -- vlasnik posta (nađen preko emaila)
    GETDATE(),                      -- vrijeme kreiranja
    0,                              -- početni score
    0,                              -- početni broj pregleda
    'Kako optimizirati SQL upite pok 2?',  -- naslov
    'Koje su najbolje prakse za optimizaciju SQL Server upita?' -- sadržaj
FROM Users U
WHERE U.EmailHash = 'ABC123XYZ';
```

Naredna stored procedura je pokrenuta više puta i korištena sa istim filterima kao prethodni AD HOC upit.

```
CREATE PROCEDURE InsertNewPost
    @EmailHash NVARCHAR(100),
    @Title NVARCHAR(250),
    @Body NVARCHAR(MAX)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @UserId INT;

    -- pronađemo korisnika preko EmailHash
    SELECT TOP 1 @UserId = Id
    FROM Users
    WHERE EmailHash = @EmailHash;

    -- ako je korisnik pronađen, ubacujemo post
    IF @UserId IS NOT NULL
    BEGIN
        INSERT INTO Posts (PostTypeId, OwnerUserId, CreationDate, Score, ViewCount, Title, Body)
        VALUES (1, @UserId, GETDATE(), 0, 0, @Title, @Body);

        PRINT 'Post uspješno dodat.';
    END
    ELSE
    BEGIN
        PRINT 'Nije pronađen korisnik sa zadanim EmailHash.';
    END
END
GO

EXEC InsertNewPost
    @EmailHash = 'ABC123XYZ',
    @Title = 'Kako optimizirati SQL upite pok 2?',
    @Body = 'Koje su najbolje prakse za optimizaciju SQL Server upita?';
```

4.3.3. Test H1_Q3_SP3

H1 Označava hipotezu 1, Q3 označava upit 3, dok SP1 označava stored procedure 3.

Objave (Posts) su nasoj bazi podataka su nastale od 2008-07-31 21:42:52.667 do 2013-12-31 23:59:48.203. Naredni AD HOC upit je pokrenut direktno u SSMS. Ovaj upit ažurira sve postove napravljene prije 2009 (vjerojatno najmanji broj postova jer baza ne obuhvata čitavu 2009. godinu). i povećava njihov ViewCount za 10 * broj godina starosti.

```
UPDATE Posts
SET ViewCount = ViewCount + DATEDIFF(YEAR, CreationDate, GETDATE()) * 10
WHERE CreationDate < '2009-01-01';
```

Naredna je stored procedura u istom stilu kao i prethodni AD HOC upit.

```
USE StackOverflow2013;
GO   -- <<< ovo razdvaja batch
CREATE PROCEDURE UpdateOldPostsViewCount
    @YearCutoff DATE,
    @Multiplier INT
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE Posts
    SET ViewCount = ViewCount + DATEDIFF(YEAR, CreationDate, GETDATE()) * @Multiplier
    WHERE CreationDate < @YearCutoff;
END
GO

EXEC UpdateOldPostsViewCount
    @YearCutoff = '2009-01-01',
    @Multiplier = 10;
```

4.3.4. Test H1_Q4_SP4

H1 Označava hipotezu 1, Q4 označava upit 4, dok SP1 označava stored procedure 4. U testu 2 sam dodavala postove u bazu, sad ću ih obrisati. Pri testiranju opet ću pokretati isti insert, istih postova, te ih opet isto brisati. Prvo sam obrisala sve dodane, a zatim ću dodavati 1, brisati jedan, akako bi pri testiranju svi rezultati bili nad istim brojem redova, konstantno.

```
DELETE FROM Posts
WHERE Title = 'Kako optimizirati SQL upite pok 2?';
```

Ispod se nalazi ekvivalentna Stored procedure.

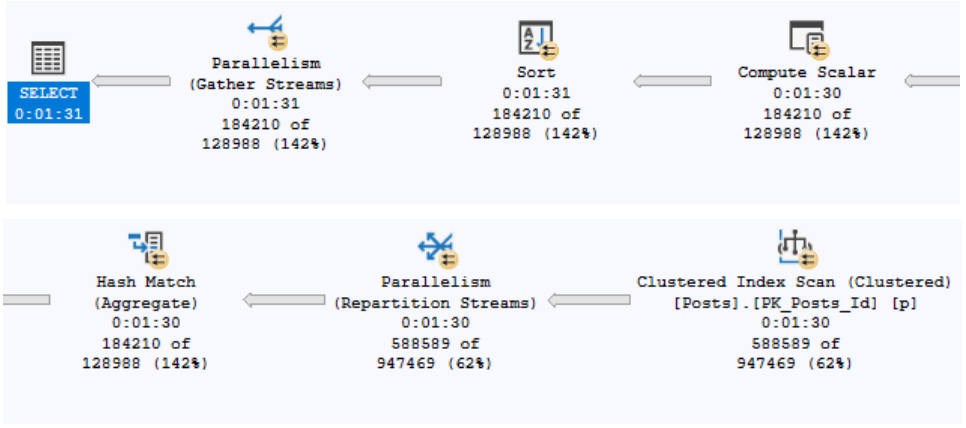
```
CREATE PROCEDURE DeletePostByTitle
    @Title NVARCHAR(250)
AS
BEGIN
    SET NOCOUNT ON;

    DELETE FROM Posts
    WHERE Title = @Title;
END
GO

EXEC DeletePostByTitle @Title = 'Kako optimizirati SQL upite pok 2?';
```

4.4. Rezultati testiranja

4.4.1. Rezultat H1_Q1_SP1

Rezultati AD HOC upita:

Što se tiče performansi ovaj ad hoc upit se pokazao prilično sporim trajao je u prosjeku oko 92 sekunde i zahtijevao skeniranje cijele Posts tabele. To znači da SQL Server troši puno resursa (CPU + IO) da bi došao do rezultata, jer nema optimizovan indeks niti unaprijed spremljeni plan. Očekuje se da će performanse biti stabilnije i prosječno brže kod Stored procedure zbog keširanja execution plana.



Slike 3. Live Query Statistics pri prvom pokretanju

Pokretanje 1: 93994 ms (~94 s)

Pokretanje 2: 92019 ms (~92 s)

Pokretanje 3: 91282 ms (~91 s)

Pokretanje 4: 92310 ms (~92 s)

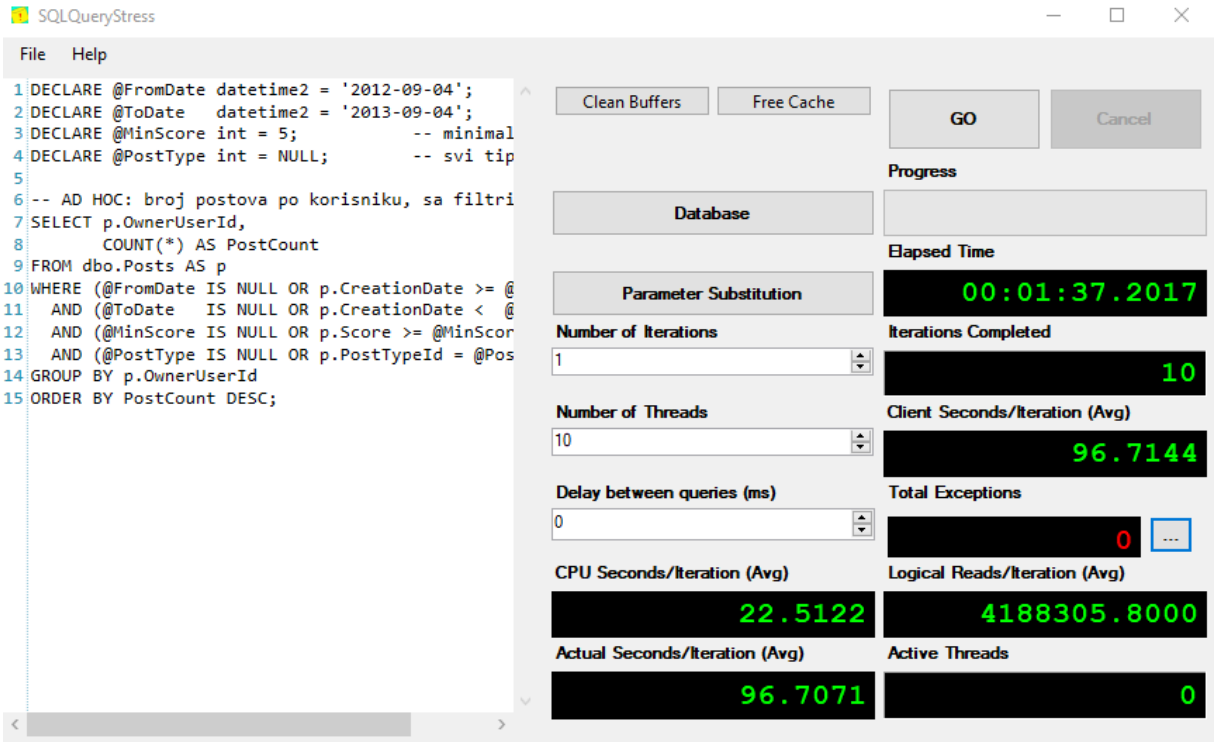
Pokretanje 5:90910 ms (~91 s)

Srednje trajanje jednog izvršavanja: 92013 ms (~92 s)

	Trial 1	Average
Client Execution Time	14:56:29	
Query Profile Statistics		
Number of INSERT, DELETE and UPDATE statements	0 →	0.0000
Rows affected by INSERT, DELETE, or UPDATE stateme...	0 →	0.0000
Number of SELECT statements	7 →	7.0000
Rows returned by SELECT statements	184222 →	184222.0000
Number of transactions	0 →	0.0000
Network Statistics		
Number of server roundtrips	3 →	3.0000
TDS packets sent from client	3 →	3.0000
TDS packets received from server	262 →	262.0000
Bytes sent from client	1470 →	1470.0000
Bytes received from server	2076068 →	2076068.0000
Time Statistics		
Client processing time	225 →	225.0000
Total execution time	93994 →	93994.0000
Wait time on server replies	93769 →	93769.0000

Slika 4. Client Statistics pri prvom izvršavanju

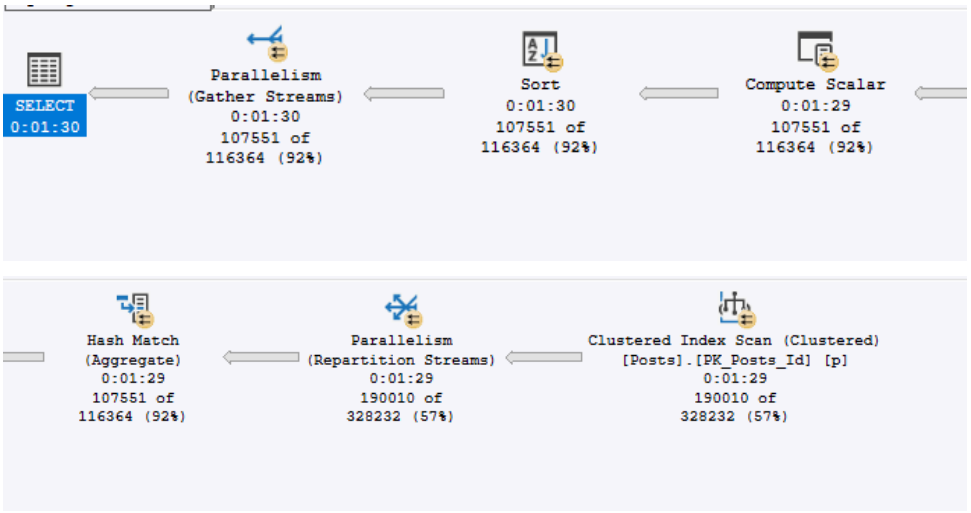
Koristila sam SQLQueryStress za testiranje šta se dešava ukoliko više klijenata pošalje zahtjev za izvršavanje upita u isto vrijeme. Koristila sam čak mali broj klijenata- samo 10 i razlika je već uveliko vidljiva. Prosječno vrijeme izvršavanja poraste na čak 96.71 s, što je skoro 5 sekundi duže nego kad samo jedan klijent šalje zahtjev.



Slika 5. SQLQueryStress test sa 10 klijenata

Rezultati Stored procedure:

Vrlo korisno pri pokretanju Stored Procedure jeste što Client Statistics odmah izvrši inicijalnih 7 pokretanja, te je lahko uraditi pregled i dobiti gotovo srednje trajanje jednog izvršavanja.



Slike 6 i 7. Live Query Statistics pri prvom pokretanju

	Trial 7	Trial 6	Trial 5	Trial 4	Trial 3	Trial 2	Trial 1	Average
Client Execution Time	15:30:22	15:27:56	15:25:17	15:14:20	15:11:38	15:09:03	14:56:29	
Query Profile Statistics								
Number of INSERT, DELETE and UPDATE statements	0 →	0 →	0 →	0 →	0 →	0 →	0 →	0.0000
Rows affected by INSERT, DELETE, or UPDATE stateme...	0 →	0 →	0 →	0 →	0 →	0 →	0 →	0.0000
Number of SELECT statements	3 ↑	0 →	0 →	7 ↓	7 →	7 →	7 →	4.4286
Rows returned by SELECT statements	0 →	0 →	0 →	184222 ↓	184222 →	184222 →	184222 →	105269.7000
Number of transactions	0 →	0 →	0 →	0 →	0 →	0 →	0 →	0.0000
Network Statistics								
Number of server roundtrips	3 →	3 →	3 →	3 →	3 →	3 →	3 →	3.0000
TDS packets sent from client	3 →	3 →	3 →	3 →	3 →	3 →	3 →	3.0000
TDS packets received from server	156 ↑	3 →	3 →	262 ↓	262 →	262 →	262 →	172.8571
Bytes sent from client	524 ↓	1426 →	1426 ↓	1470 →	1470 →	1470 →	1470 →	1322.2860
Bytes received from server	1231744 ↑	115 →	115 ↓	2076045 ↓	2076183 ↑	2076043 ↓	2076068 →	1362330.0000
Time Statistics								
Client processing time	107 ↑	0 →	0 ↓	233 ↓	295 ↑	290 ↑	225 →	164.2857
Total execution time	89915 ↑	15 ↓	31 ↓	89795 ↓	89809 ↓	90045 ↓	93994 →	64800.5700
Wait time on server replies	89808 ↑	15 ↓	31 ↓	89562 ↑	89514 ↓	89755 ↓	93769 →	64636.2900

Slika 8. Client Statistics pri prvom izvršavanju

Na slici 7 je vidljivo 7 pokretanja upita koristeći Stored Procedure (Trial 1 → Trial 7). SSMS je za svako izvođenje mjerio mrežu, broj redova, execution time itd.

Gleda se, kao i prethodno “Total execution time” red i tu imamo sljedeće rezultate:

Trial 1- 93994 ms (~94 s)

Trial 2- 90045 ms(~90 s)

Trial 3- 89089 ms(~89 s)

Trial 4- 89795 ms(~90 s)

Trial 5- 89964 ms(~90 s)

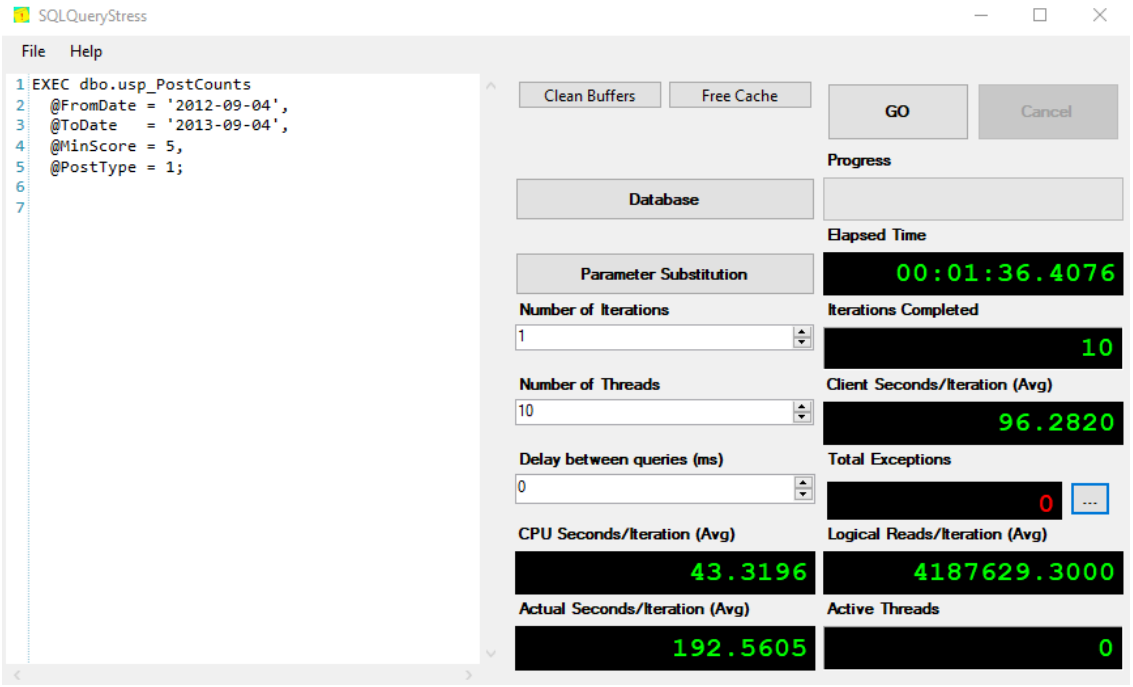
Trial 6- 89900 ms(~90 s)

Trial 7- 89,915 ms(~90 s)

Dakle, prvo pokretanje je bilo najsporije, a od drugog nadalje se vrijeme stabilizovalo na 89-90 sekundi. Ovo potvrđuje jednim dijelom postavljenu hipotezu. Pretpostavka je bila da će trebati manje vremena za izvršavanje upravo radi razloga što se jedan execution plan više puta koristi i ne kreira se uvijek.

Pri prvom izvođenju SQL Server mora da uradi parsing, optimizaciju i kreiranje execution plana.Također učitava podatke iz diska u keš memoriju.Zato je sporije.

Za Trial 2 i dalje execution plan je već spremljen u plan cache. Podaci su već dobrim dijelom u buffer cache-u (RAM-u, a ne disku). Zato su kasnija izvođenja brža i stabilnija.



Slika 9. SQLQueryStress test sa 10 klijenata

Koristila sam SQLQueryStress za testiranje šta se dešava ukoliko više klijenata pošalje zahtjev za izvršavanje upita u isto vrijeme. Koristila sam čak mali broj klijenata- samo 10 i razlika je već uveliko vidljiva. Prosječno vrijeme izvršavanja poraste na čak 96.28s, što je skoro pa jednako slanju zahtjeva AD HOC načinom.

4.4.2. Rezultat H1_Q2_SP2

Nakon izrade aplikacije koristim isključivo nju za testiranje hipoteza, uvjerila sam se testiranjem na upitima sa Testa H1_Q1_SP1 da daje skoro pa iste rezultate kao i sam SSMS. Rad same aplikacije će biti pojašnjen u nekom drugom poglavlju, ovo i daljnja poglavlja ‘rezultati’ će sadržati rezultate iz moje aplikacije, te po potrebi neke dodatne statistike iz SSMS.

Rezultati AD HOC upita:

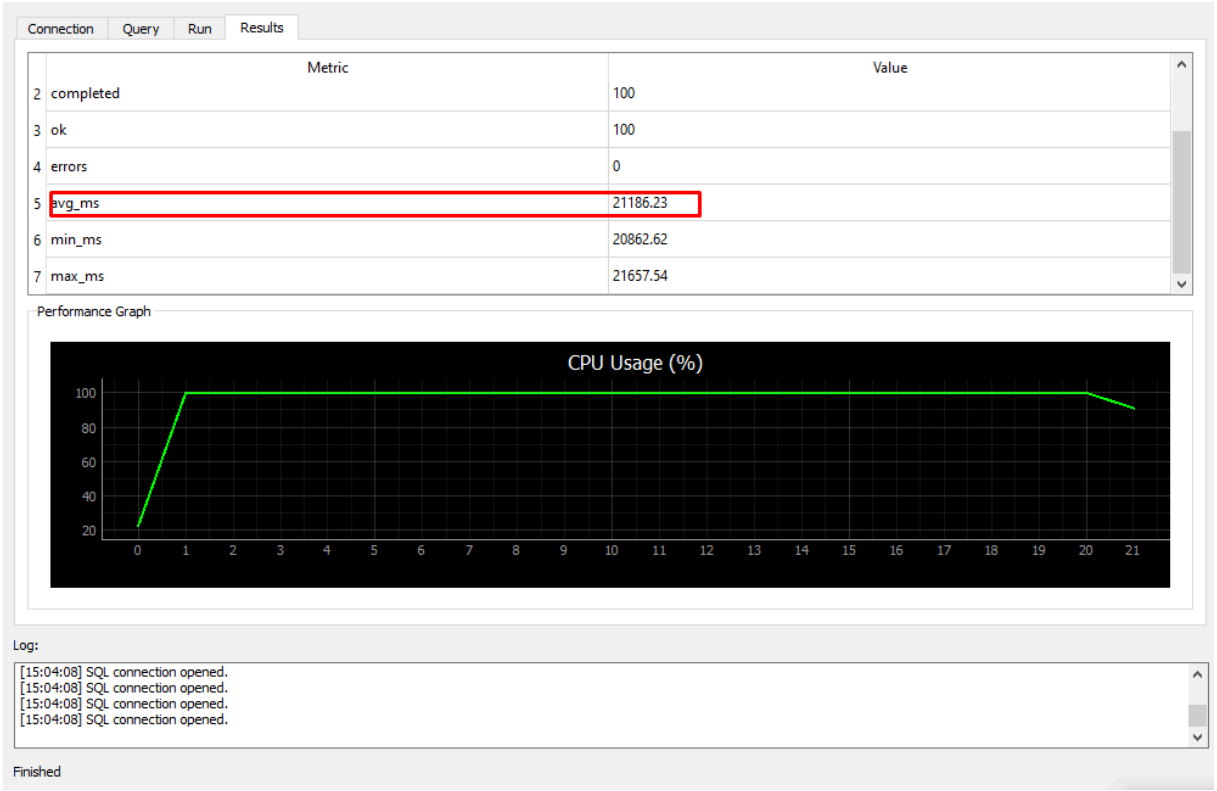
Za samo obavljanje ovog upita je potrebno vrlo malo vremena, jer za raliku od prvog, nema potrebe da prolazi kroz velik broj zapisa, već samo pristupa tabeli i upisuje jedan. Vrijeme izvršavanja upita je sve ispod 0.3 sekunde, što se vidi kroz 7 Triala u tabeli Client statistics ispod.

	Trial 8	Trial 7	Trial 6	Trial 5	Trial 4	Trial 3	Trial 2	Trial 1	Average
Client Execution Time	14:50:00	14:39:55	14:39:34	14:39:17	14:38:51	14:38:22	14:38:04	14:31:15	
Query Profile Statistics									
Number of INSERT, DELETE and UPDATE statements	2 ↓	4 ↑	0 →	0 →	0 ↓	4 ↑	0 ↓	2 →	1.5000
Rows affected by INSERT, DELETE, or UPDATE stateme...	0 ↓	1 ↑	0 →	0 →	0 ↓	1 ↑	0 →	0 →	0.2500
Number of SELECT statements	2 ↓	6 ↑	0 →	0 →	0 ↓	6 ↑	0 ↓	2 →	2.0000
Rows returned by SELECT statements	8 ↓	15 ↑	0 →	0 →	0 ↓	15 ↑	0 ↓	8 →	5.7500
Number of transactions	2 ↓	4 ↑	0 →	0 →	0 ↓	4 ↑	0 ↓	2 →	1.5000
Network Statistics									
Number of server roundtrips	3 →	3 →	3 →	3 →	3 →	3 →	3 →	3 →	3.0000
TDS packets sent from client	3 →	3 →	3 →	3 →	3 →	3 →	3 →	3 →	3.0000
TDS packets received from server	8 →	8 ↑	3 →	3 →	3 ↓	8 ↑	3 ↓	8 →	5.5000
Bytes sent from client	1458 ↑	374 ↓	1522 ↑	1520 ↑	330 ↓	372 ↓	1584 ↑	1462 →	1077.7500
Bytes received from server	47576 ↑	44625 ↑	115 ↓	306 ↓	343 ↓	44807 ↑	115 ↓	47574 →	23182.6300
Time Statistics									
Client processing time	20 ↓	232 ↑	0 →	0 →	0 ↓	238 ↑	0 →	0 →	61.2500
Total execution time	392 ↑	247 ↑	0 ↓	37 ↑	0 ↓	253 ↑	15 ↓	377 →	165.1250
Wait time on server replies	372 ↑	15 ↑	0 ↓	37 ↑	0 ↓	15 →	15 ↓	377 →	103.8750

Slika 10. Client statistics za ovaj AD HOC upit

Prosjek je 165 ms .To je ukupno vrijeme, tu ulazi i obrada na serveru + mrežni delay + klijent obrada. Kreće se od 247 ms do 37 ms u nekim trialovima.

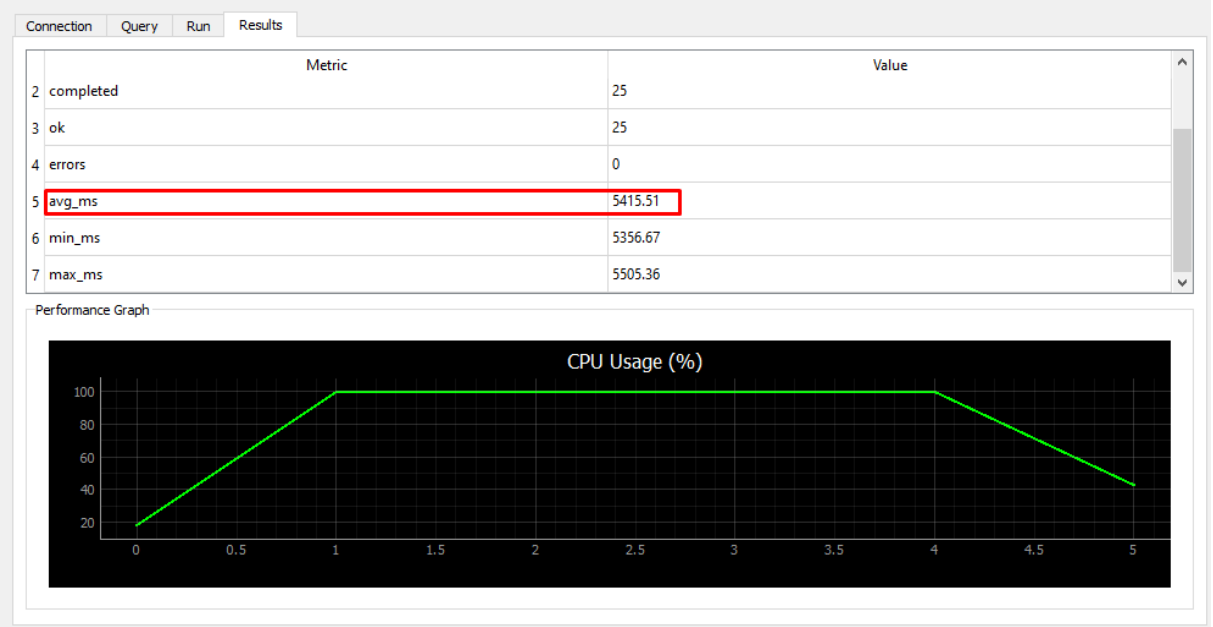
S obzirom na jednostavnost ovog upita vrijeme trajanja istog je testirano u mojoj aplikaciji na 100 klijenata koji šalju zahtjev u isto vrijeme.



Slika 11. Rezultati testa preko vlastite aplikacije

Za odziv u slučaju 100 klijenata koji šalju zahtjev u tačno isto vrijeme je potrebno oko 21.2 sekunde, a korištenje resursa CPU je na 100% gotovo svo vrijeme.

Malo realističniji slučaj je da 25 klijenata pošalje zahtjev, naprimjer u razmaku od po 1s. U tom slučaju dobijamo rezultat da je za odziv potrebno u prosjeku 5.4s. U jednom trenutku korištenje resursa CPU raste na 100%. Prikazano na slici ispod.



Slika 12. Rezultati testa preko vlastite aplikacije

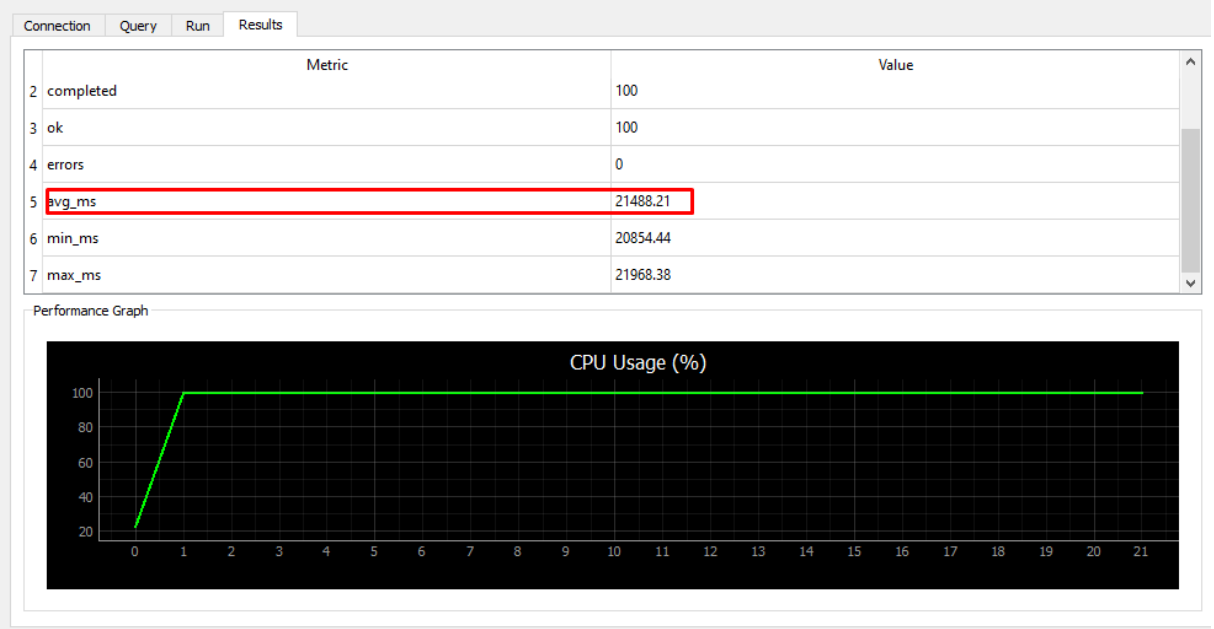
Rezultati Stored procedure:

Za samo kreiranje stored procedure je potrebno ispod 1s, a za obavljanje upita nad njom je potrebno vrlo malo vremena. Vrijeme izvršavanja upita je u prosjeku 169ms, što je čak duže nego za AD HOC upit. Ovo može ukazati na to da nike uvijek isplatno raditi stored procedure, tj. da je možda isplatno samo kad se velik set podataka mora čitav 'proći'.

	Trial 9	Trial 8	Trial 7	Trial 6	Trial 5	Trial 4	Trial 3	Trial 2	Trial 1	Average
Client Execution Time	15:06:21	14:50:00	14:39:55	14:39:34	14:39:17	14:38:51	14:38:22	14:38:04	14:31:15	
Query Profile Statistics										
Number of INSERT, DELETE and UPDATE statements	4 ↑	2 ↓	4 ↑	0 →	0 →	0 ↓	4 ↑	0 ↓	2 →	1.7778
Rows affected by INSERT, DELETE, or UPDATE stateme...	1 ↑	0 ↓	1 ↑	0 →	0 →	0 ↓	1 ↑	0 →	0 →	0.3333
Number of SELECT statements	6 ↑	2 ↓	6 ↑	0 →	0 →	0 ↓	6 ↑	0 ↓	2 →	2.4444
Rows returned by SELECT statements	15 ↑	8 ↓	15 ↑	0 →	0 →	0 ↓	15 ↑	0 ↓	8 →	6.7778
Number of transactions	4 ↑	2 ↓	4 ↑	0 →	0 →	0 ↓	4 ↑	0 ↓	2 →	1.7778
Network Statistics										
Number of server roundtrips	3 →	3 →	3 →	3 →	3 →	3 →	3 →	3 →	3 →	3.0000
TDS packets sent from client	3 →	3 →	3 →	3 →	3 →	3 →	3 →	3 →	3 →	3.0000
TDS packets received from server	8 →	8 →	8 ↑	3 →	3 →	3 ↓	8 ↑	3 ↓	8 →	5.7778
Bytes sent from client	374 ↓	1458 ↑	374 ↓	1522 ↑	1520 ↑	330 ↓	372 ↓	1584 ↑	1462 →	999.5555
Bytes received from server	44629 ↓	47576 ↑	44625 ↑	115 ↓	306 ↓	343 ↓	44807 ↑	115 ↓	47574 →	25565.5500
Time Statistics										
Client processing time	199 ↑	20 ↓	232 ↑	0 →	0 →	0 ↓	238 ↑	0 →	0 →	76.5556
Total execution time	206 ↓	392 ↑	247 ↑	0 ↓	37 ↑	0 ↓	253 ↑	15 ↓	377 →	169.6667
Wait time on server replies	7 ↓	372 ↑	15 ↑	0 ↓	37 ↑	0 ↓	15 →	15 ↓	377 →	93.1111

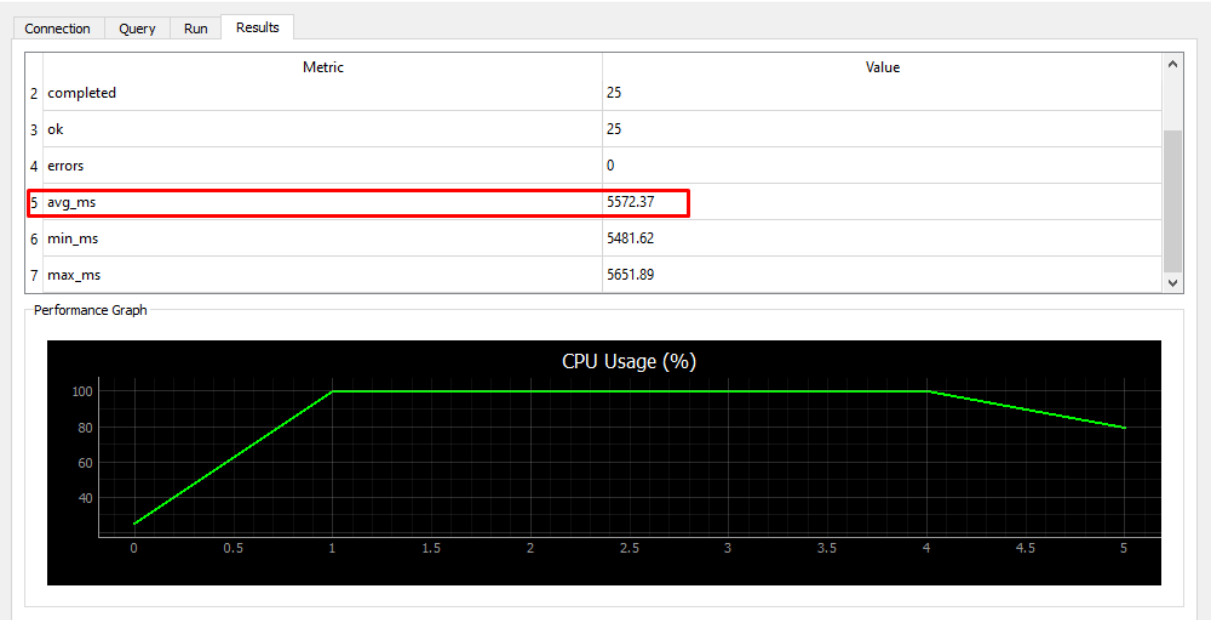
Slika 13. Client statistics za ovaj Stored Procedure

Dalje je testirano nad 100 klijenata, koji svi u isto vrijeme šalju zahtjev. U slučaju da 100 klijenata šalje zahtjev u tačno isto vrijeme, za select i insert je potrebno u prosjeku 21.4s, što je opet malo duže nego pri AD HOC upitu.



Slika 14. Rezultati testa preko vlastite aplikacije

Malo realističniji slučaj je da 25 klijenata pošalje zahtjev, naprimjer u razmaku od po 1s. U tom slučaju dobijamo rezultat da je za odziv potrebno u prosjeku 5.5s. U jednom trenutku korištenje resursa CPU raste na 100%. Prikazano na slici ispod.

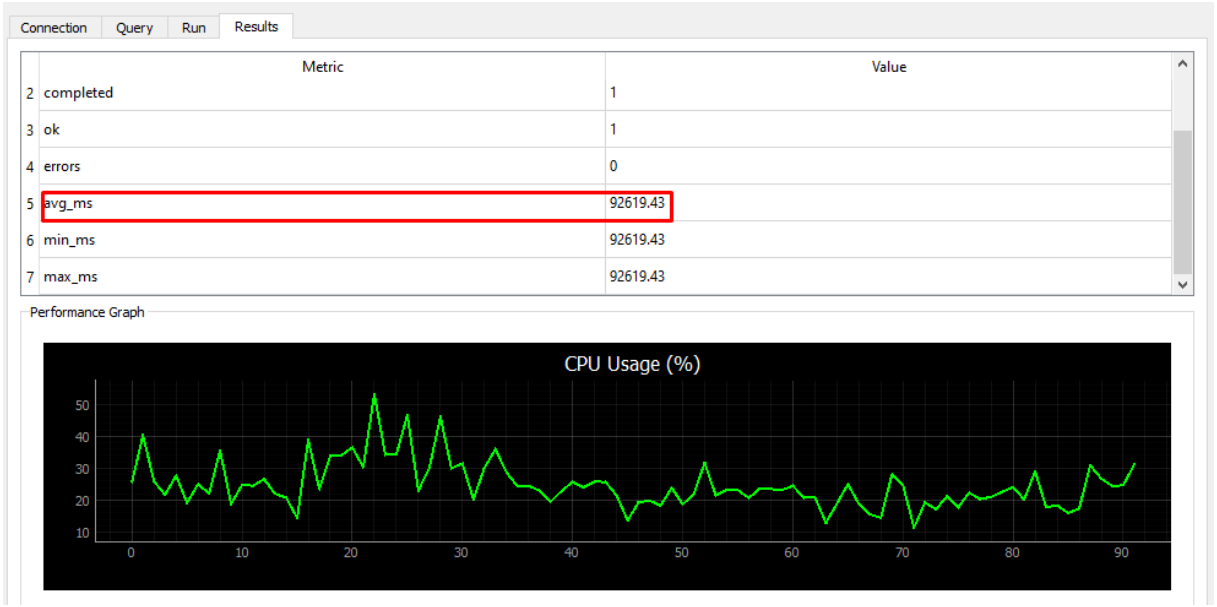


Slika 15. Rezultati testa preko vlastite aplikacije

4.4.3. Rezultat H1_Q3_SP3

Rezultati AD HOC upita:

Prosjek: izvršavanja 1 upita od strane 1 klijenta je 92620 ms (minut i 32 sekunde). Dakle, ovo UPDATE je zahtjevniji jer zahvata veliki broj redova i radi izračune (DATEDIFF). Na testiranju putem aplikacije je moguće vidjeti i opterećenje CPU tokom izvršavanja ovog upita, a ono se kreće do 50% u jednom trenutku.

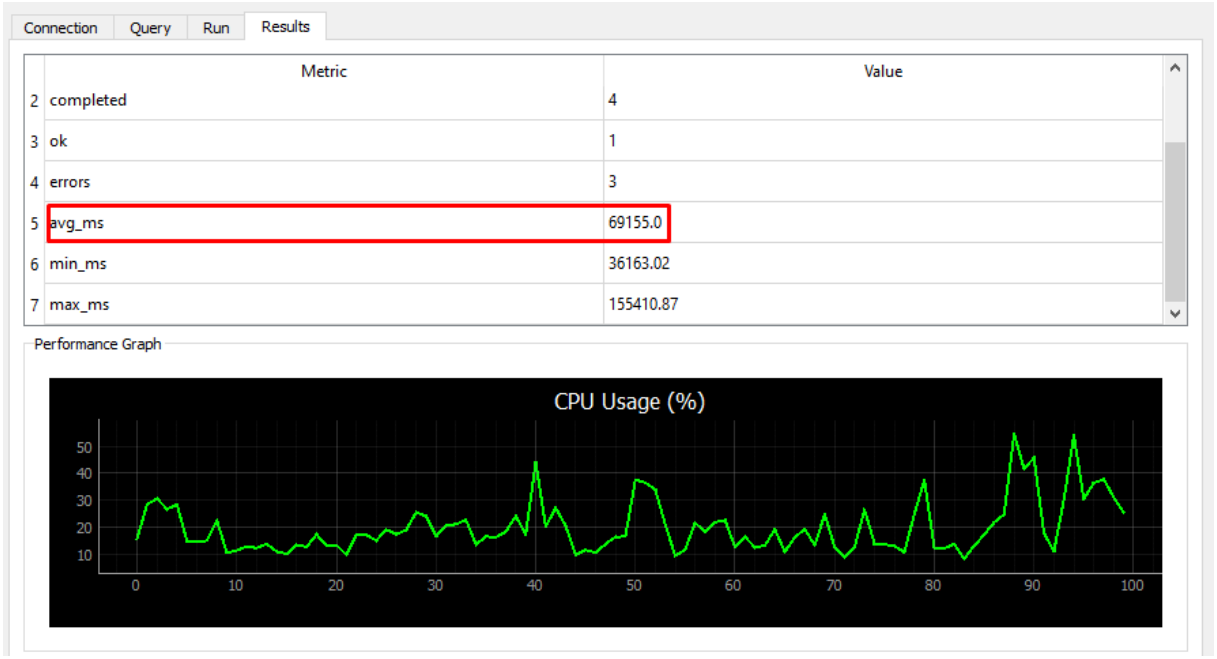


Slika 16. Rezultati testa preko vlastite aplikacije

Mnogo zahtjevniji primjer koji će biti korišten i za stored procedure je 4 klijenata u isto vrijeme (ali parametar su svi postovi nastali 2011. godine, dakle puno je više postova, što još više 'podiže' kompleksnost upita). Opterećenost CPU dostiže do 50%.

Samo čekanje u prosjeku traje oko 69s, ali bitno je zapaziti da minimalno vrijeme i maksimalno vrijeme mnogo odudaraju jedno od drugog. Minimalno vrijeme (vidljivo na slici ispod) je oko 36s, dakle ispod minute. Maksimalno vrijeme je oko 155s, dakle oko nekih 2 i po minute.

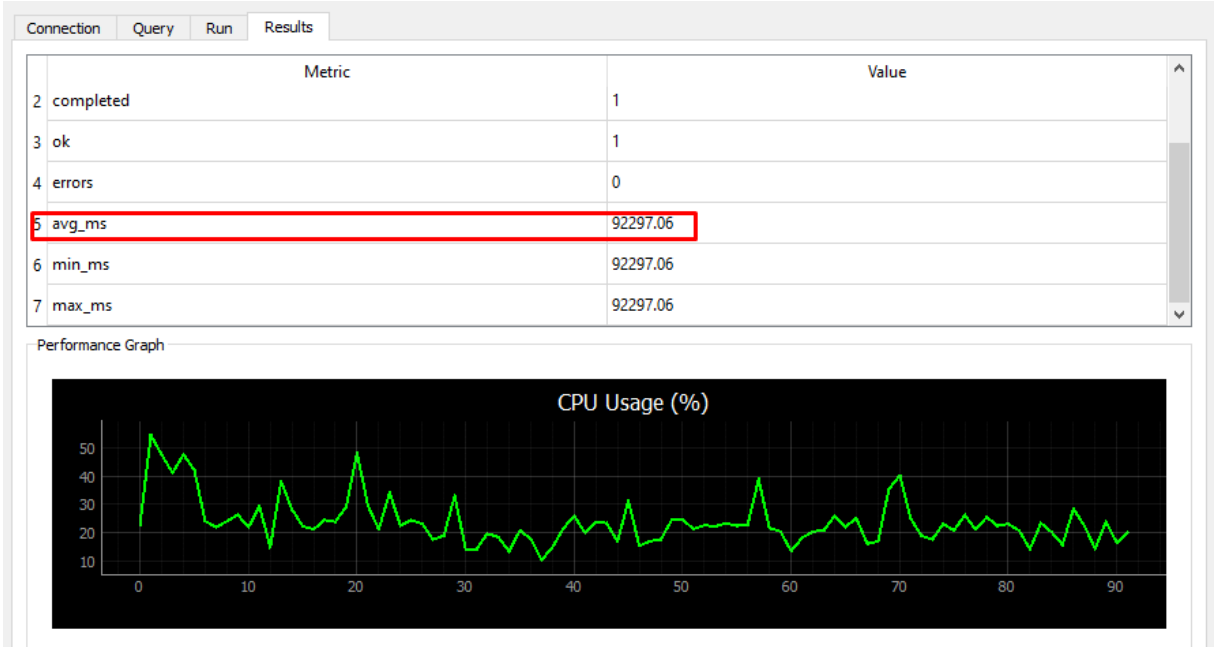
Bitno je napomenuti da je na slici ispod prikazano opterećenje CPU samo zadnjih 100s , radi postavki grafa unutar aplikacije. Ovo jeste jedan od propusta, ali je također razlog zašto aplikacija analizira opterećenje koje nastaje upitom, i vrijeme čekanja odgovora, te ne stvara sama od sebe dodatno opterećenje i ne iziskiva dodatno vrijeme za obradu.



Slika 16. Rezultati testa preko vlastite aplikacije

Rezultati Stored procedure:

Prosjek: izvršavanja 1 upita od strane 1 klijenta je 92297 ms (minut i 29 sekundI). Opterećenje CPU tokom izvršavanja ovog upita, a ono se kreće do 50% u jednom trenutku. Ovo je za neke 3 sekunde brže nego izvršavanje putem AD HOC upita. Ovo je prikazano naravno na slici ispod



Slika 17. Rezultati testa preko vlastite aplikacije

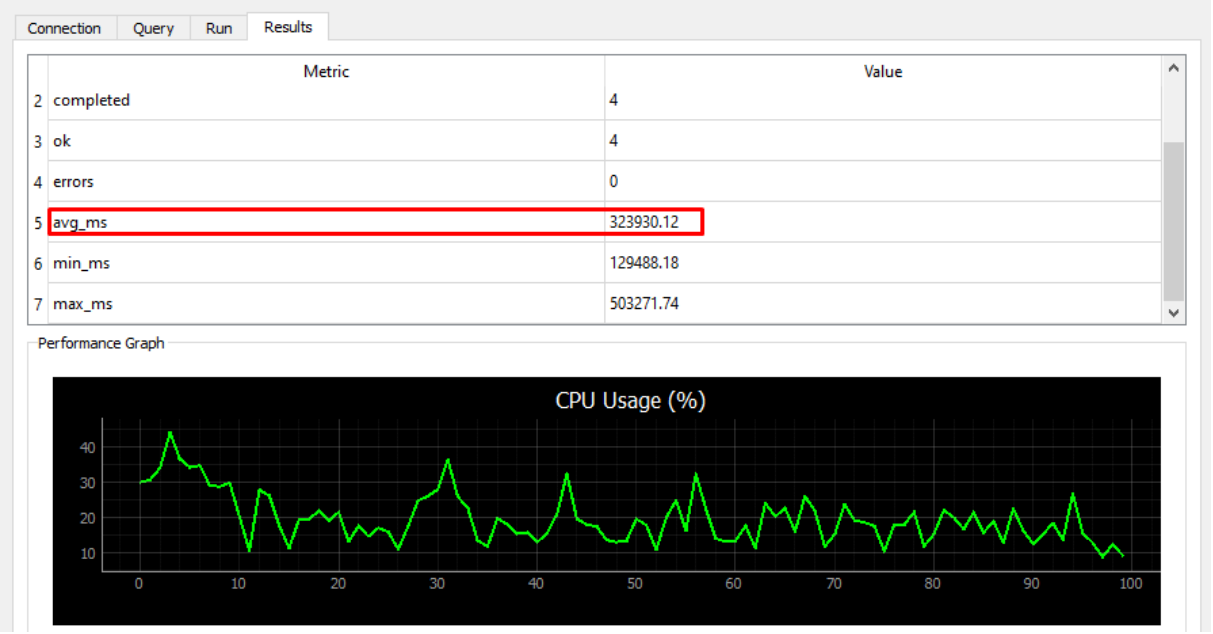
Mnogo zahtejvniji primjer koji će biti korišten i za stored procedure je 4 klijenata u isto vrijeme (ali parametar su svi postovi nastali 2011. godine, dakle puno je više postova, što još

više ‘podiže’ kompleksnost upita). Opterećenost CPU dostiže do 50%. Da bih ovo uradila bila je potrebna izmjena same Stored procedure, prikazana ispod.

```
USE StackOverflow2013;
GO   -- <<< ovo razdvaja batch
ALTER PROCEDURE UpdateOldPostsViewCount
    @YearFrom DATE = '2011-01-01',
    @YearTo   DATE = '2011-12-31',
    @Multiplier INT
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE Posts
    SET ViewCount = ViewCount + DATEDIFF(YEAR, CreationDate, GETDATE()) * @Multiplier
    WHERE CreationDate BETWEEN @YearFrom AND @YearTo;
END
GO
```

Ovdje sam dobila puno losije rezultate nego za ad hoc upit. Provjerila sam i 2 puta izvršila pokretanje procedure. Pri jednom sam dobila prosječno vrijeme oko 3 i po minute, što je i dlaje lošije nego za AD HOC. Na slici 18 ću prikazati prvobitno, puno gore pokretanje. Samo čekanje u prosjeku traje oko 323s, dakle 5 puta više nego za AD HOC upit. Što se tiče minimalnog i maksimalnog vremena , oba su otprilike u omjeru 5 puta veći nego ona za AD HOC upit.

Bitno je napomenuti da je na slici ispod prikazano opterećenje CPU samo zadnjih 100s , radi postavki grafa unutar aplikacije.



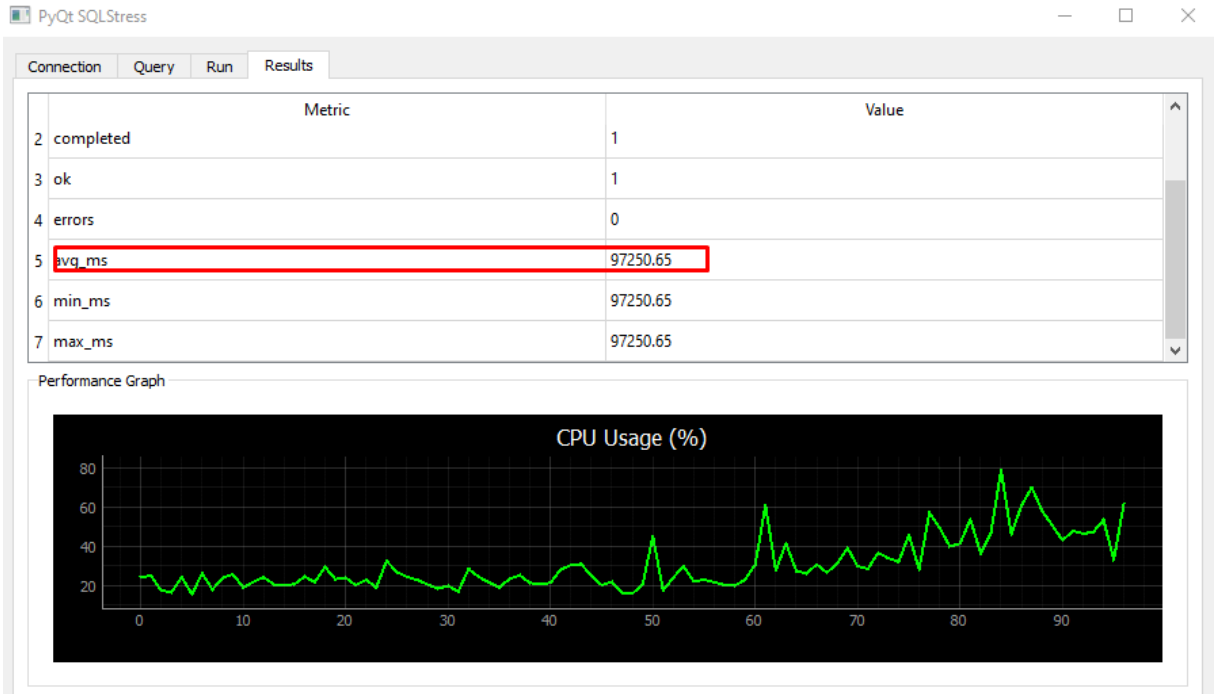
Slika 18. Rezultati testa preko vlastite aplikacije

4.4.4. Rezultat H1_Q4_SP4

Rezultati AD HOC upita:

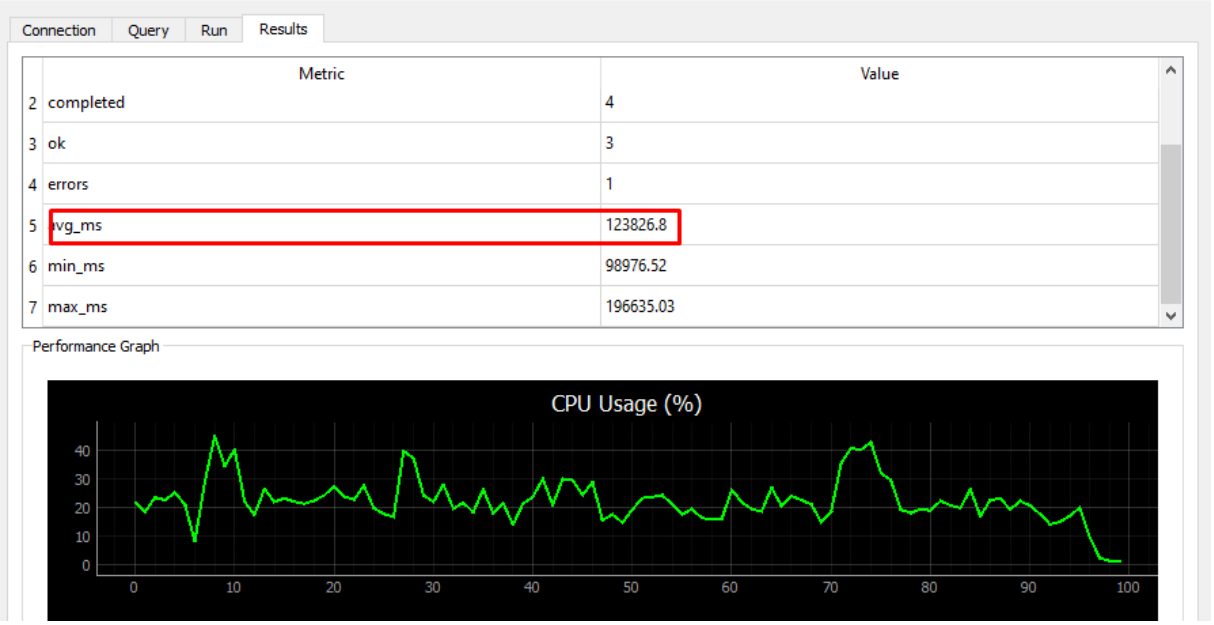
Radi se DELETE FROM Posts WHERE Title = '...', a Title kolona nije indeksirana , pa SQL mora napraviti full table scan (pročitati sve redove i tražiti one koje treba obrisati). Što je tabela veća, to traje duže. Za slučaj 1 korisnika i 1 zahtjeva, dobija se trajanje u prosjeku od 97s, a CPU resursi se koriste čak do 80%, dakle ovo je mnogo zahtjevnije nego svi prethodni upiti.

Dalje sam ubacila opet post u bazu, te testirala izvršavanje sa Stored procedurom (1 klient, 1 zahtjev).



Slika 19. Rezultati testa preko vlastite aplikacije

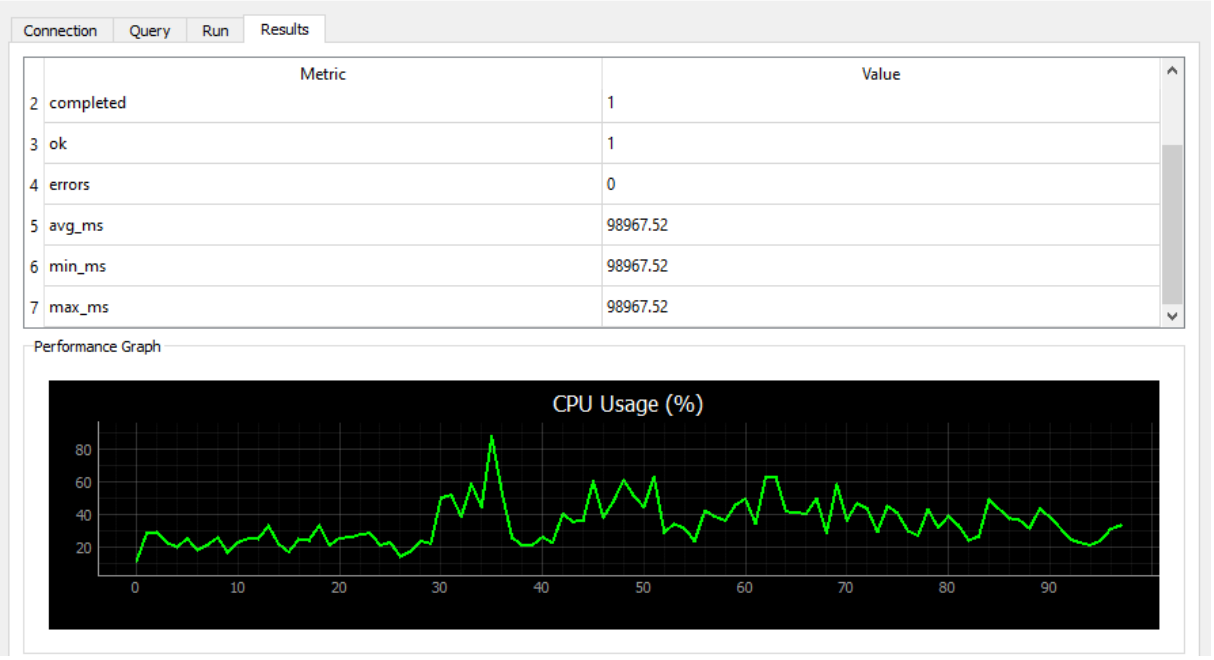
Radi ove minimalne razlike između AD HOC i stored procedure poziva odlučila sam testirati šta se dešava kada više klijenata pristupa istoj stvari, tj, ovdje je najveći problem pretraživanje po neindeksiranoj koloni. Odlučila sam testirati slučaj kada 2 klijenta šalju po 2 zahtjeva, sa razmakom od 5s. Koliko vremena je potrebno za odziv? Rezultat je zanimljiv, u prosjeku 123s za izvršenje upita, najkraće vrijeme je 90s, a najduže 196s. Ono što je zanimljivo jeste kako korištenje CPU opada- samo 40% resursa je iskorišteno.



Slika 20. Rezultati testa preko vlastite aplikacije

Rezultati Stored procedure:

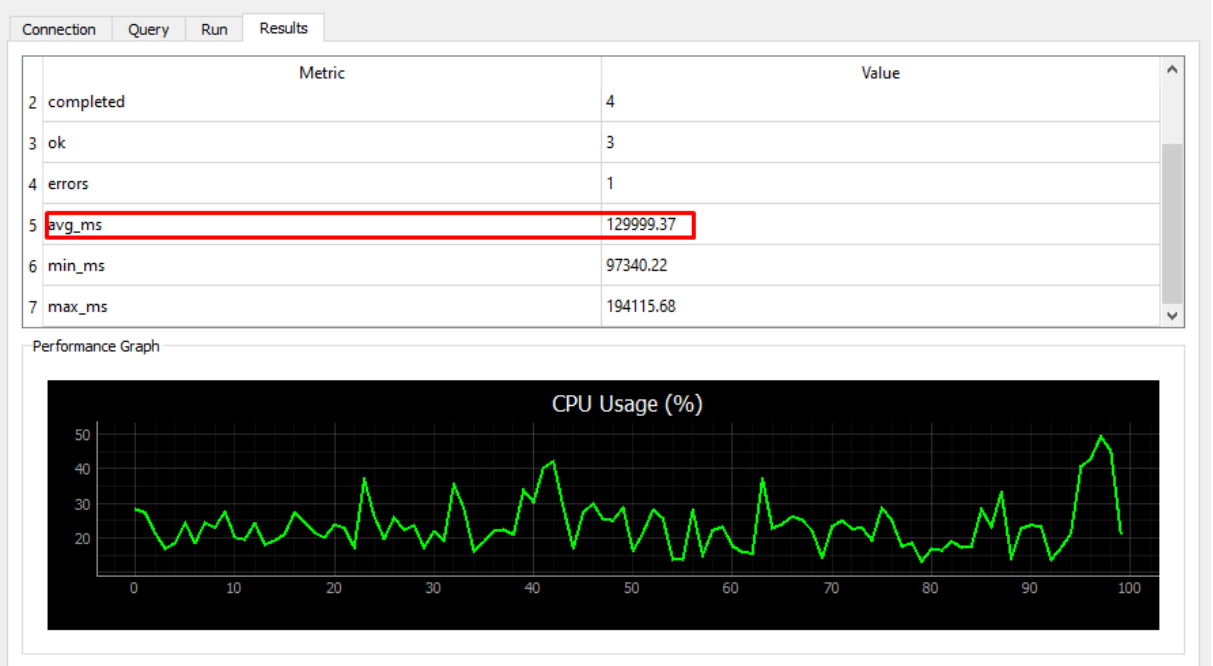
Za jednog klijenta, jedan zahtjev, su performanse gotovo pa iste- Stored Procedure doduše zahtjeva 1 sekundu duže no AD HOC zahtjev. Upotrebljenost CPU je ista. Radi ove minimalne razlike odlučila sam testirati šta se dešava kada više klijenata pristupa istoj stvari, tj, ovdje je najveći problem pretraživanje po neindeksiranoj koloni.



Slika 21. Rezultati testa preko vlastite aplikacije

Za 2 klijenta, koji šalju po 2 zahtjeva, u razmaku od po 5s, rezultat je sljedeći: prosječno vrijeme odziva 129s (oko 2 minute). Minimalno vrijeme odziva 97s a maksimalno vrijeme odziva 194s (oko 3 minute). Korištenje CPU doseže i do 80%, kao i za AD HOC upit. Nisu

velike razlike od AD HOC upita, ali doista, nisu ni bolje performanse. Svako od ovih vremena sem maksimalnog, traje za par sekundi duže od čistog AD HOC upita.



Slika 22. Rezultati testa preko vlastite aplikacije

4.5. Konačni zaključak za Hipotezu 1

Hipoteza 1 se **nije potvrdila**.

Stored procedure nisu u svim slučajevima imale bolje performanse od AD HOC upita. Pokazale su blagu prednost samo kod ponavljanih SELECT upita, dok su kod INSERT, UPDATE i DELETE operacija razlike bile zanemarive ili čak u korist ad hoc pristupa. Dakle, hipoteza se u cjelini smatra **opovrgnutom**.

Nakon svega ću napraviti kratki osvrt na rezultat svakog testiranja:

- H1_Q1_SP1- Select upit. Prosječno izvršavanje AD HOC upita traje preko 92s. Prosječno izvršavanje pomoću Stored procedure jeste oko 90s. To je 2s razlike. Potrebno je uzeti u obzir da se ovdje radi o 1 klijentu i slanju 1 zahtjeva. Ukoliko 10 klijenata šalje zahtjev prosječno izvršavanje AD HOC upita je preko 96 sekundi, a stored procedure je skoro pa isti, s obzirom na to povećan je još više broj korisnika (na 100), te se pokazalo da je vrijeme izvršavanja za obe situacije predugo.

  Ovo se može prepisati tome što se prolazi kroz čitave tabele, nije urađena nikakva optimizacija povodom čitanja podataka, već samo načina na koji se sama procedura čitanja obavlja. Te se može reći da u slučaju SELECT naredbe Stored procedure jeste brži, ali s brojem klijenata se razlika među dva pristupa umanjuje.

U tabeli ispod se nalazi pregled samo prosječnih vremena i maksimalna opterećenost CPU.[1]

H1_Q1_SP1					
Broj korisnika	Broj iteracija	Q (t + CPU)		SP (t + CPU)	
1	1	92013 ms		90276ms	
10	1	96710 ms		96280ms	

Tabela 1. Prikaz rezultata testa H1_Q1_SP1

- H1_Q2_SP2- Select+ insert upit. Prosječno izvršavanje AD HOC upita (INSERT sa SELECT podupitom) trajalo je oko 165 ms, što je vrlo kratko jer se ne radi o velikom setu podataka nego o jednostavnom unosu novog reda. Kod testiranja sa 100 klijenata koji šalju zahtjev istovremeno, vrijeme odziva se povećalo na oko 21.2 sekunde, uz konstantno 100% opterećenje CPU-a. Kada je 25 klijenata slalo zahtjeve u razmaku od po 1 sekundu, prosječno vrijeme odziva iznosilo je oko 5.4 sekunde.

  Rezultati za Stored procedure pokazali su prosječno trajanje od 169 ms, što je čak malo duže nego kod AD HOC upita. Iako se stored procedure u pravilu isplate kod složenijih operacija nad velikim setovima podataka (gdje se execution plan ponovo koristi), ovdje zbog jednostavnosti samog INSERT-a nema značajne koristi. Testovi sa više klijenata (100 istovremeno i 25 sukcesivno) također pokazuju gotovo identične vrijednosti kao kod AD HOC pristupa, pa se može zaključiti da za jednostavne DML operacije (INSERT) prednost Stored procedure praktično ne dolazi do izražaja.

H1_Q2_SP2					
Broj korisnika	Broj iteracija	Q (t + CPU)		SP (t + CPU)	
1	1	247ms	100%	169ms	100%
100	1	21185 ms	100%	21488ms	100%
25	1 /1s	5415 ms	100%	5572ms	100%

Tabela 2. Prikaz rezultata testa H1_Q2_SP2

- H1_Q3_SP3- Update upit. Kod AD HOC upita, prosječno vrijeme izvršavanja jednog zahtjeva od strane jednog klijenta bilo je oko 92.6 sekundi. CPU opterećenje je uočeno do 50% tokom izvršavanj. Za Stored procedure, prosjek izvršavanja jednog zahtjeva od strane jednog klijenta bio je gotovo isti.

  Pri testiranju sa 4 klijenta istovremeno (nad svim postovima iz 2011. godine) dobivena su vrlo varijabilna vremena od minimalno 36 sekundi pa sve do maksimalno 155 sekundi, što ukazuje na osjetljivost performansi u zavisnosti od trenutnog opterećenja servera. U ovom scenariju, za stored procedure sa 4 klijenta dobiveni su čak lošiji rezultati nego kod AD HOC-a: prosjek se kretao oko 3.5 minute, dok je u najlošijem

[1] Ovo važi za svaku tabelu u radu.

slučaju dostigao i 323 sekunde (~5 minuta). To pokazuje da stored procedure ne donose uvijek prednost kod složenih i masovnih UPDATE operacija.

H1_Q3_SP3					
Broj korisnika	Broj iteracija	Q (t + CPU)		SP (t + CPU)	
1	1	92620 ms	50%	92297ms	50%
4	1	69155 ms	50%	323930ms	40%

Tabela 3. Prikaz rezultata testa H1_Q3_SP3

- H1_Q4_SP4- Delete upit. Glavni razlog za dugo trajanje je činjenica da se brisanje vrši po koloni Title, koja nije indeksirana, pa SQL Server mora napraviti full table scan i proći kroz cijelu tabelu da bi pronašao odgovarajuće redove. Rezultati za Stored procedure pokazali su gotovo identične performanse kao i AD HOC upit. U scenariju sa jednim klijentom razlika u vremenu je bila tek oko 1 sekunde (stored procedure su čak bile malo sporije). Kada su dva klijenta slala po dva zahtjeva u razmaku od 5 sekundi, prosječno vrijeme odziva bilo je oko 129 sekundi, uz minimalno trajanje od 97 i maksimalno od 194 sekunde. CPU opterećenje kod sStored procedure je u ovom slučaju dostizalo 80%, a kod AD HOC samo 40%. Dakle, čak bolje performanse ima AD HOC upit. Ovo pokazuje da u slučaju DELETE operacija nad neindeksiranim kolonama Stored procedure ne donose nikakvu prednost u odnosu na AD HOC upite, jer je usko grlo sama pretraga i skeniranje cijele tabele.

H1_Q4_SP4					
Broj korisnika	Broj iteracija	Q (t + CPU)		SP (t + CPU)	
1	1	97250 ms	80%	98967ms	80%
2	2 /5s	123826 ms	40%	129999ms	50%

Tabela 4. Prikaz rezultata testa H1_Q4_SP4

5. HIPOTEZA 2

Korištenje denormalizovanih summary tabela poboljšava performanse agregacijskih upita.

Kada sistem često vrši izvještavanje i prebrojavanje podataka (npr. broj postova po korisniku), efikasnije je koristiti prethodno izračunate vrijednosti u tzv. summary tabelama. Umjesto skeniranja miliona redova svaki put, čitanje iz denormalizovane tabele omogućava trenutne odgovore. Ovaj pristup posebno je značajan naprimjer u mobilnim aplikacijama koje zahtijevaju brze povratne informacije s minimalnim opterećenjem baze.

Ovo je moja druga hipoteza. Testiranje se može izvesti tako da se u bazi napravi posebna summary tabela koja sadrži agregirane podatke (npr. UserId, BrojPostova). U toj tabeli vrijednosti se periodično ažuriraju (INSERT/UPDATE), a zatim se izvode agregacijski upiti direktno nad ovom manjom i jednostavnijom tabelom. Na ovaj način se eliminiše potreba za ponovnim prolaskom kroz veliku Posts tabelu pri svakom zahtjevu.

5.1. Osnovni pojmovi

- *Denormalizacija*- proces u kojem se u bazu uvode redundantni podaci (višak podataka) kako bi se ubrzalo čitanje i izvještavanje. To često znači kombinovanje više tabela u jednu ili dodavanje kolona koje sadrže unaprijed izračunate vrijednosti.
- *Summary tabela -* posebna tabela koja čuva sažetke (agregirane podatke), npr. ukupan broj postova po korisniku, broj komentara po postu, broj glasova po mjesecu. Umjesto skeniranja originalne tabele sa milionima redova, sistem pristupa summary tabeli koja sadrži samo već obrađene informacije.

Za potrebe testiranja je napravljena jedna summary tabela. Ova tabela za svakog korisnika (po UserID) čuva podatke o ukupnom broju glasova i vrsti tih glasova (samo prvih 5 kategorija iz VoteType). Mini verzija baze podataka, na kojoj radimo, ne sadrži broj upvote, downvote i slično, ali sadrži broj glasova kojima je post označen kao ‘favorite’.

```
CREATE TABLE SummaryVotes (
    UserId INT PRIMARY KEY,
    TotalVotes INT NOT NULL,
    AcceptedByOriginator INT NOT NULL,
    Upvotes INT NOT NULL,
    Downvotes INT NOT NULL,
    Offensive INT NOT NULL,
    Favorite INT NOT NULL
);
```

Nakon kreiranja tabelu je potrebno ispuniti podacima:

```
INSERT INTO SummaryVotes (UserId, TotalVotes, AcceptedByOriginator, Upvotes, Downvotes, Offensive, Favorite)
SELECT
    UserId,
    COUNT(*) AS TotalVotes,
    SUM(CASE WHEN VoteTypeId = 1 THEN 1 ELSE 0 END) AS AcceptedByOriginator,
    SUM(CASE WHEN VoteTypeId = 2 THEN 1 ELSE 0 END) AS Upvotes,
    SUM(CASE WHEN VoteTypeId = 3 THEN 1 ELSE 0 END) AS Downvotes,
    SUM(CASE WHEN VoteTypeId = 4 THEN 1 ELSE 0 END) AS Offensive,
    SUM(CASE WHEN VoteTypeId = 5 THEN 1 ELSE 0 END) AS Favorite
FROM Votes
WHERE UserId IS NOT NULL
GROUP BY UserId;
```

Na slici ispod se nalazi prvih nekoliko redova ove Summary tabele.

Results Messages

	UserId	TotalVotes	AcceptedByOriginator	Upvotes	Downvotes	Offensive	Favorite
1	-1	980	0	0	0	0	0
2	1	4	0	0	0	0	0
3	2	11	0	0	0	0	11
4	3	72	0	0	0	0	72
5	4	20	0	0	0	0	19
6	5	29	0	0	0	0	28
7	9	2	0	0	0	0	2
8	10	1	0	0	0	0	1
9	13	53	0	0	0	0	53
10	16	10	0	0	0	0	10
11	17	18	0	0	0	0	17
12	20	23	0	0	0	0	22
13	22	28	0	0	0	0	26
14	23	39	0	0	0	0	39
15	24	2	0	0	0	0	2
16	25	35	0	0	0	0	33

Slika 23. Pregled Summary tabele

- *Agregacijski upiti*- upiti koji rade nad velikim setovima podataka koristeći funkcije kao što su COUNT(), SUM(), AVG(), MIN(), MAX(). U velikim bazama oni mogu biti vrlo skupi po pitanju vremena i resursa, naročito kada se ponavljaju.

5.2. Način mjerenja performansi

Za upoređivanje performansi koristi se:

- PyQT SQLStress (uvijek)- Vlastita aplikacija koja je klon SQLQueryStress ali ima dodatne funkcionalnosti i urađena je drugim programskim jezikom.

5.3. Testiranje hipoteze

5.3.1. Test H2_Q1_ST1

Ispod se nalazi upit bez korištenja summary tabele sa SELECT operacijom. Ovaj primjer je jednostavan- zanimat će nas broj glasova (upravo informacija koju imamo u summary tabeli).

```
SELECT
    V.UserId,
    COUNT(*) AS TotalVotes,
    SUM(CASE WHEN V.VoteTypeId = 1 THEN 1 ELSE 0 END) AS AcceptedByOriginator,
    SUM(CASE WHEN V.VoteTypeId = 2 THEN 1 ELSE 0 END) AS Upvotes,
    SUM(CASE WHEN V.VoteTypeId = 3 THEN 1 ELSE 0 END) AS Downvotes,
    SUM(CASE WHEN V.VoteTypeId = 4 THEN 1 ELSE 0 END) AS Offensive,
    SUM(CASE WHEN V.VoteTypeId = 5 THEN 1 ELSE 0 END) AS Favorite
FROM Votes V
WHERE V.UserId IS NOT NULL
GROUP BY V.UserId;
```

Ista stvar ali urađena nad summary tabelom:

```
SELECT
    UserId,
    TotalVotes,
    AcceptedByOriginator,
    Upvotes,
    Downvotes,
    Offensive,
    Favorite
FROM SummaryVotes;
```

5.3.2. Test H2_Q2_ST2

Ispod se nalazi upit bez korištenja summary tabele sa INSERT operacijom. Ubacuju se dodatni upvote glasovi za korisnika sa ID 49.

```
INSERT INTO Votes (PostId, UserId, VoteTypeId, CreationDate)
VALUES (12345, 49, 2, GETDATE());
```

Ista stvar ali urađena nad summary tabelom:

```
IF EXISTS (SELECT 1 FROM SummaryVotes WHERE UserId = 49)
BEGIN
    UPDATE SummaryVotes
    SET
        TotalVotes = TotalVotes + 1,
        Upvotes = Upvotes + 1  -- jer je VoteTypeId = 2
    WHERE UserId = 49;
END
ELSE
BEGIN
    INSERT INTO SummaryVotes (UserId, TotalVotes, AcceptedByOriginator, Upvotes, Downvotes, Offensive, Favorite)
    VALUES (49, 1, 0, 1, 0, 0, 0);
END
```

5.3.3. Test H2_Q3_ST3

Ispod se nalazi upit bez korištenja summary tabele. U njemu promijenimo tip glasa za određenog korisnika (recimo UserId = 49) iz Downvote (VoteTypeId = 3) u Upvote (VoteTypeId = 2):

```
UPDATE TOP (1) Votes
SET VoteTypeId = 2
WHERE UserId = 49 AND VoteTypeId = 3;
```

Sad se isto radi na Summary tabeli, samo malo drugi način:

```
UPDATE SummaryVotes
SET
    Downvotes = Downvotes - 1,
    Upvotes = Upvotes + 1
WHERE UserId = 49;
```

5.3.4. Test H2_Q4_ST4

Naredni test će biti jednostavan upit ali kompleksna stvar. Radi se o brisanju zapisa 'Favorite' glasanja iz čitave baze. Favorite glas je izabran zato što je on jedini popunjen u ovoj mini verziji baze, sad će se i on svesti na 0. Ispod se nalazi kako bi upit izgledao bez summary tabele.

```
DELETE FROM Votes
WHERE VoteTypeId = 5;
```

Naredni je upit iste prirode ali nad summary tabelom SummaryVotes:

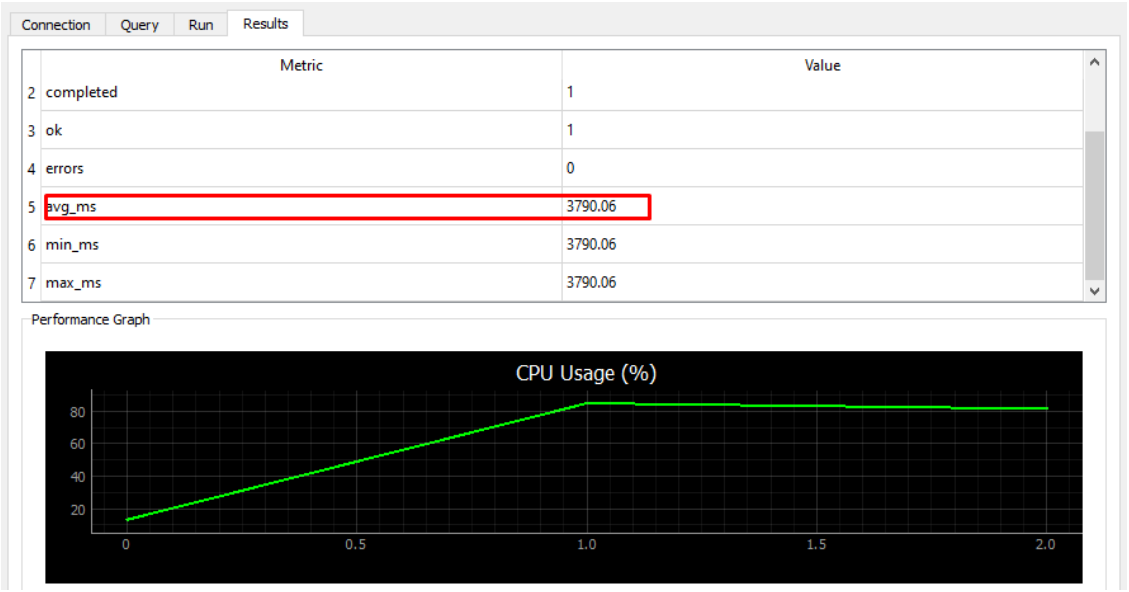
```
UPDATE SummaryVotes
SET
    Favorite = 0;
```

5.4. Rezultati testiranja

5.4.1. Rezultat H2_Q1_ST1

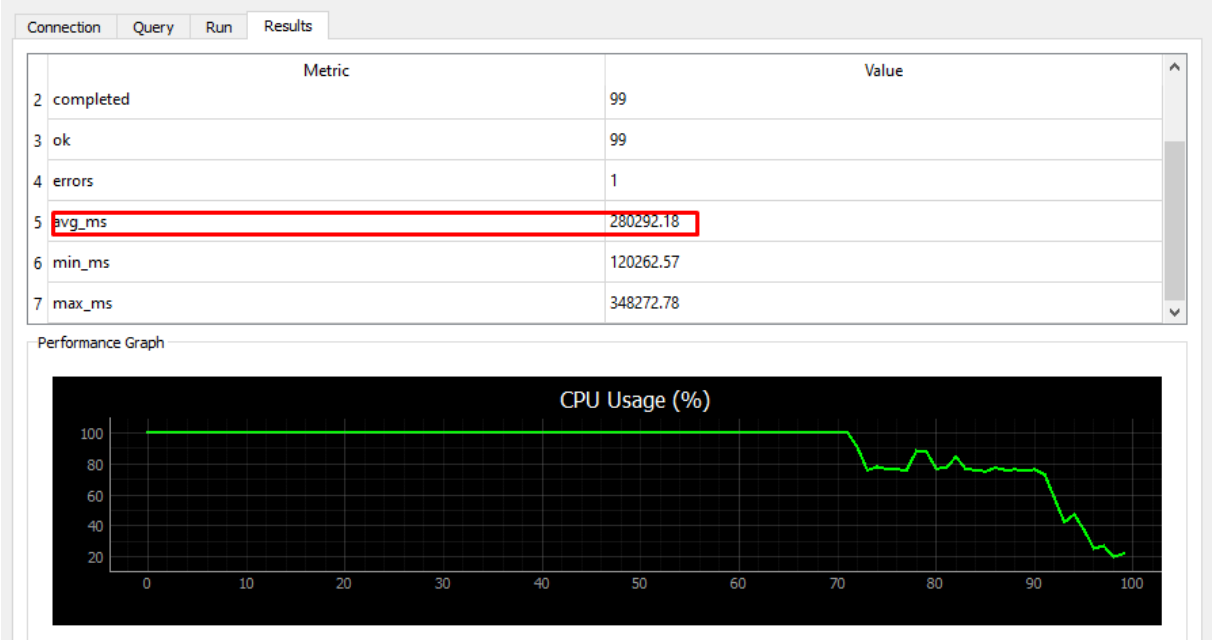
Rezultat bez summary tabele:

Testiran je upit bez summary tabele nad 1 korisnikom i jednim zahtjevom. Za odgovor je bilo potrebno oko 3.8 sekundi, to jeste 3790 milisekundi a opterećenost CPU je dosegla nekih 80% u jednom trenutku.
Iako se razlika među radom sa i bez summary tabele odma primjetila pri ovom testu, naredno je testirano šta će se desiti kada 100 korisnika pošalje zahtjev, u razmaku od po 0.1s, što bi za neku veću aplikaciju mogao biti realističan slučaj.



Slika 24. Rezultati testa preko vlastite aplikacije

Iako se razlika među radom sa i bez summary tabele odma primjetila pri ovom testu, naredno je testirano šta će se desiti kada 100 korisnika pošalje zahtjev, u razmaku od po 0.1s, što bi za neku veću aplikaciju mogao biti realističan slučaj. Na slici ispod se nalaze rezultati.

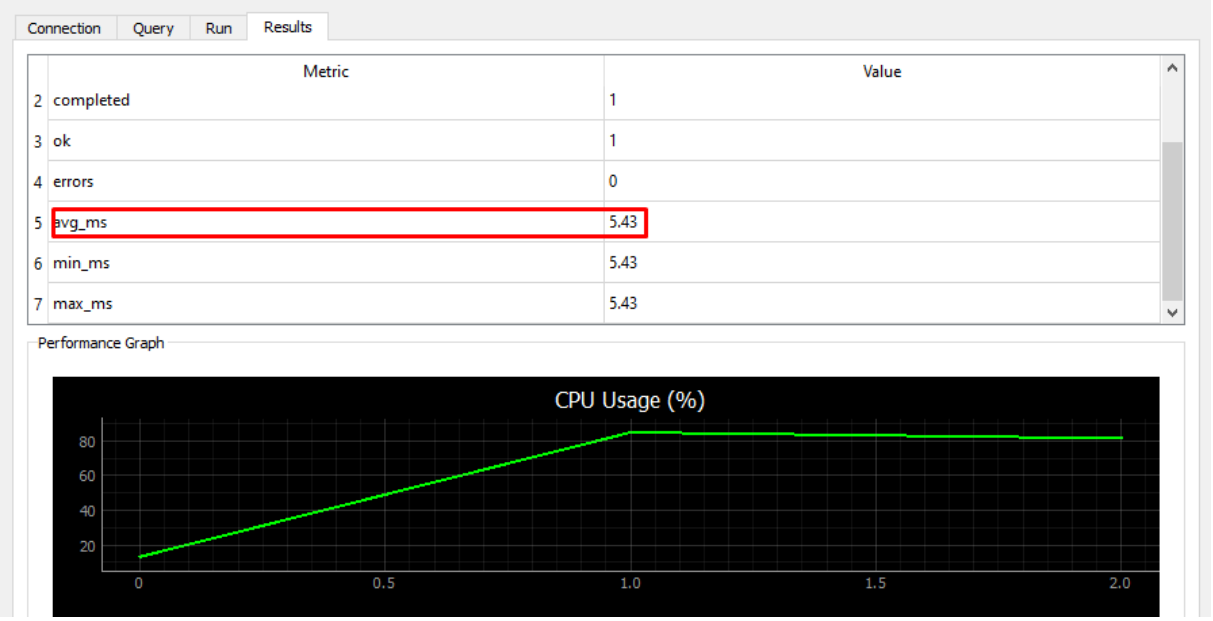


Slika 25. Rezultati testa preko vlastite aplikacije

Kao što je vidljivo, umeđuvremenu se desio problem sa serverom, ali je bitno napomnuti da aplikacija radi na način da vrijeme trajanja testa nike isto što i vrijeme odziva na upit, tako da taj problem nike uticao na same rezultate. Prosječno vrijeme izvršavanje je oko 280929ms ili 280 sekundi (4 minute i 40s). Najkraće vrijeme izvršavanja je 120s, a najduže 348s. Isti ovaj test je zatim ponovljen nad summary tabelom.

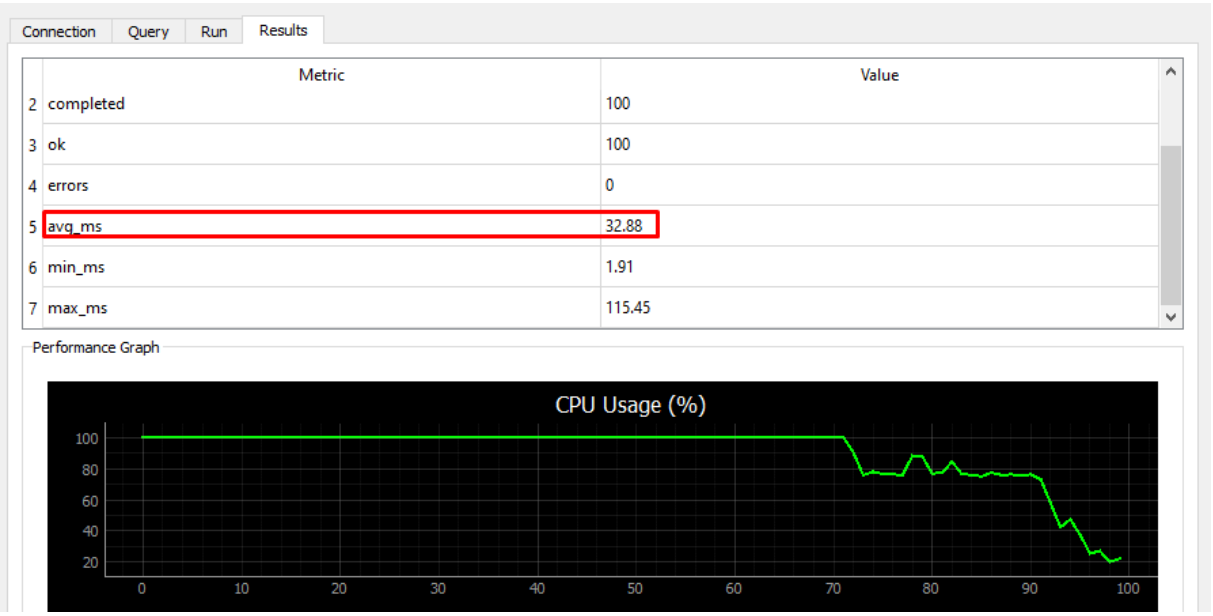
Rezultat sa summary tabelom:

Ista stvar kao prethodno je testirana nad summary tabelom. Slučaj jednog korisnika i jednog zahtjeva. Za izvršavanje ovog upita je bilo potrebno samo 5 milisekundi, što je oko 742 puta brže nego bez summary tabele. Opterećenost CPU je ista, ali je brzina izvršavanja puno veća.



Slika 26. Rezultati testa preko vlastite aplikacije

Naredni je bio test sa 100 korisnika koji šalju zahtjeve u razmaku od 0.1s. Rezultati su zanimljivi. Prosječno vrijeme odziva je bilo 32ms, minimalno vrijeme 2ms, a maksimalno vrijeme 115ms. Ovo je značajno vrijeme , prosječno vrijeme je oko 8 hiljada kraće nego prosječno vrijeme za rad bez summary tabele. Možemo reći da za SELECT upit, summary tabela daje iznimno dobre rezultate i dokazuje postavljenu hipotezu.

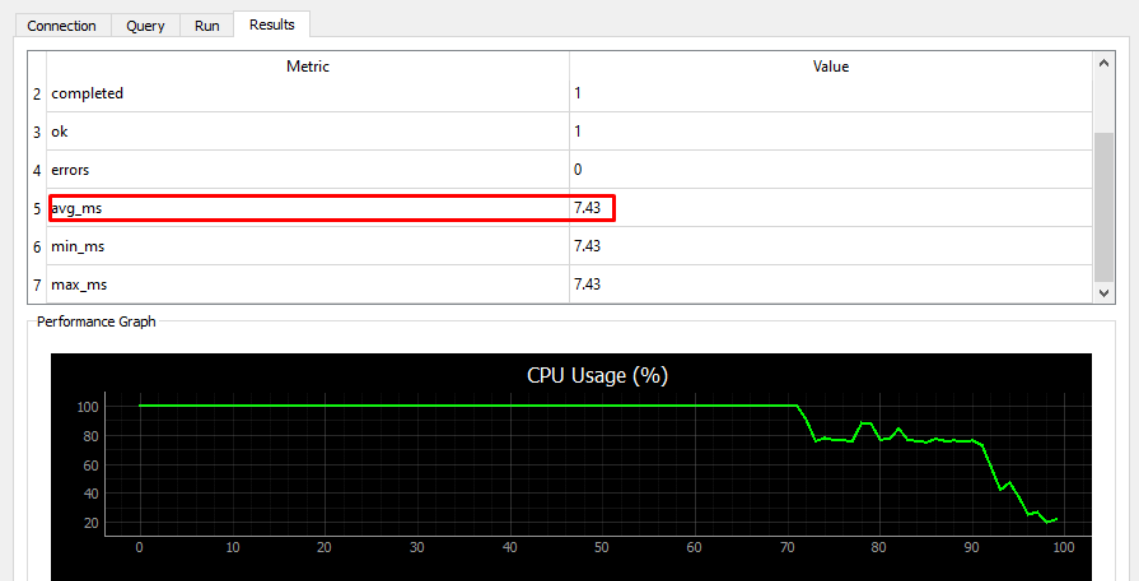


Slika 27. Rezultati testa preko vlastite aplikacije

5.4.2. Rezultat H2_Q2_ST2

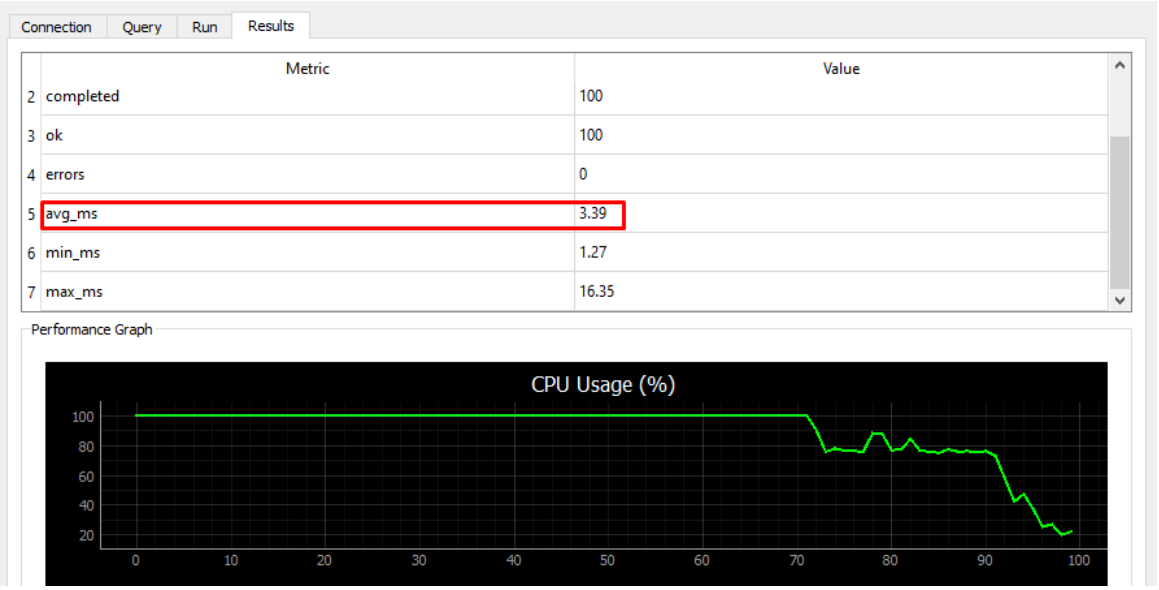
Rezultat bez Summary tabele:

Za jednog korisnika koji šalje jedan zahtjev, vrijeme izvršavanja ovog inserta je bezazleno i traje oko 7.5ms. Zbog jednostavnosti ovog upita isti je naredno testiran nad 100 korisnika koji šalju zahtjev u potpuno isto vrijemes.



Slika 28. Rezultati testa preko vlastite aplikacije

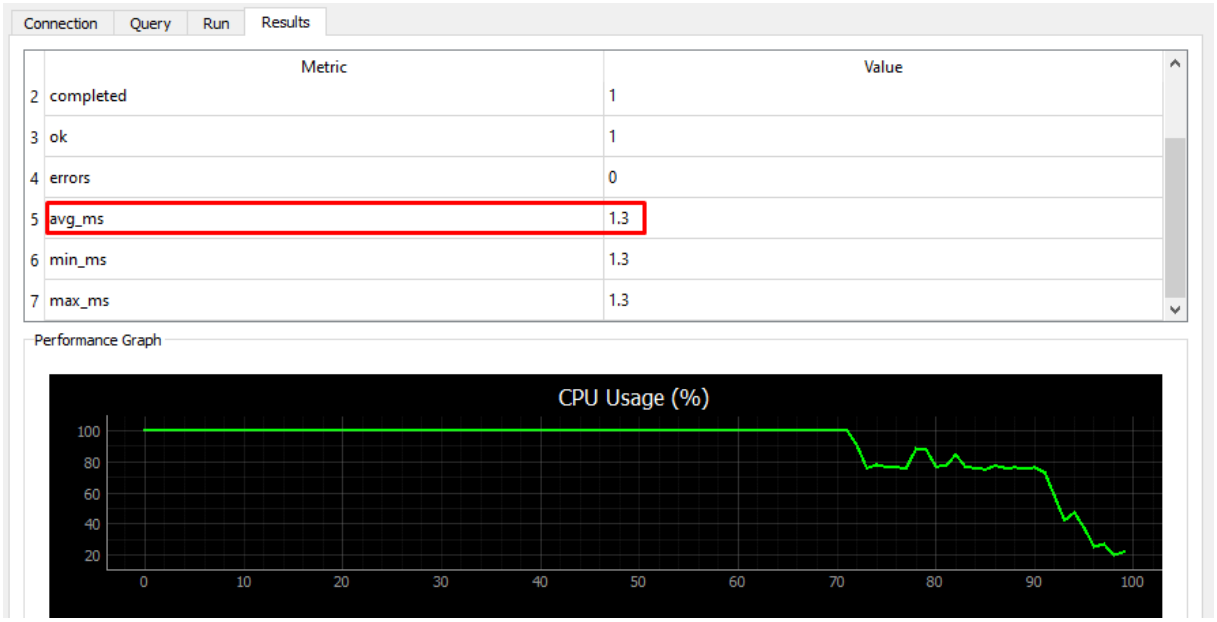
Pri ispitivanju 100 korisnika koji šalju zahtjev u isto vrijeme.Dobiveni rezultati upučuju na jednostavnost ovog upita. Prosječno virjeme odziva bi trajalo oko 3.5ms, najkraće vrijeme odziva 1.27s a najduže oko 16s, sva ova vremena trajanja su zanemariva.



Slika 29. Rezultati testa preko vlastite aplikacije

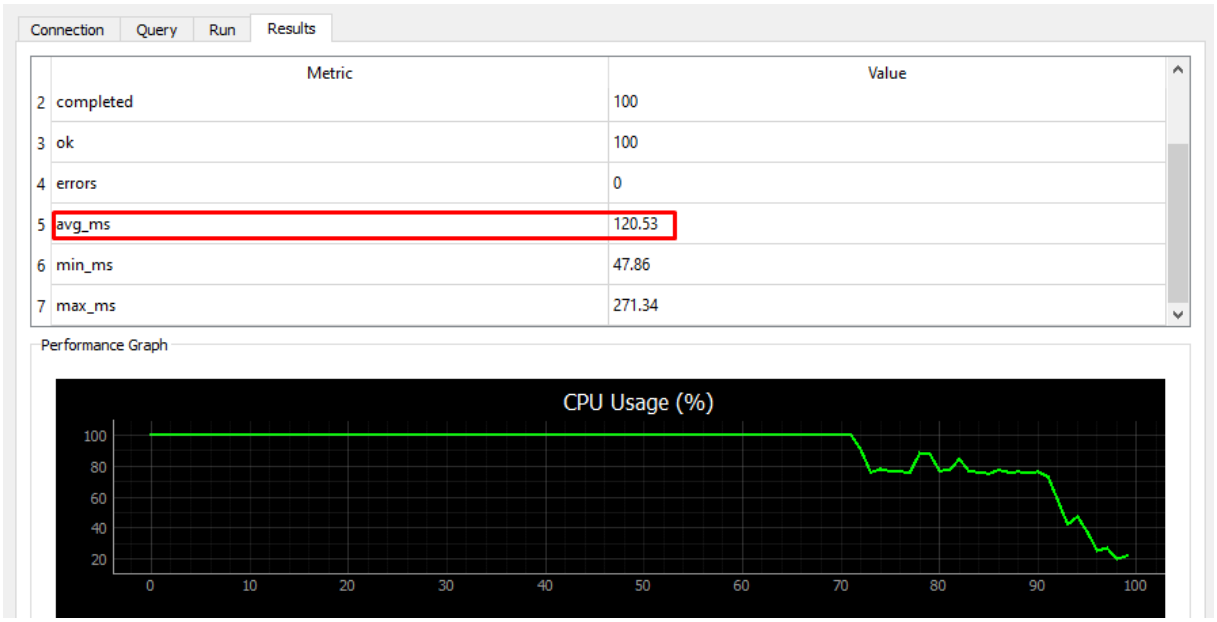
Rezultat sa Summary tabelom:

Kada je testirano sa summary tabelom, slučaj 1 korisnika i 1 zahtjeva,vrijeme odziva je i dalje kratko i zanemarivo i traje 1.2ms, što je 5 puta brže od onog bez korištenja summary tabele.



Slika 30. Rezultati testa preko vlastite aplikacije

Naredno je testirano nad 100 korisnika koji šalju zahtjev u potpuno isto vrijeme. Prosječno vrijeme trajanja odziva je 120ms, što je 40puta duže nego bez summary tabele,ali je i dalje zanemarivo vrijeme. Dakle, ovdje ispaltivost (na velikom broju korisnika) nike dokazana, ali ni opovrgnuta, s obzirom da su vremena oba izvršavanja ispod pola sekunde.

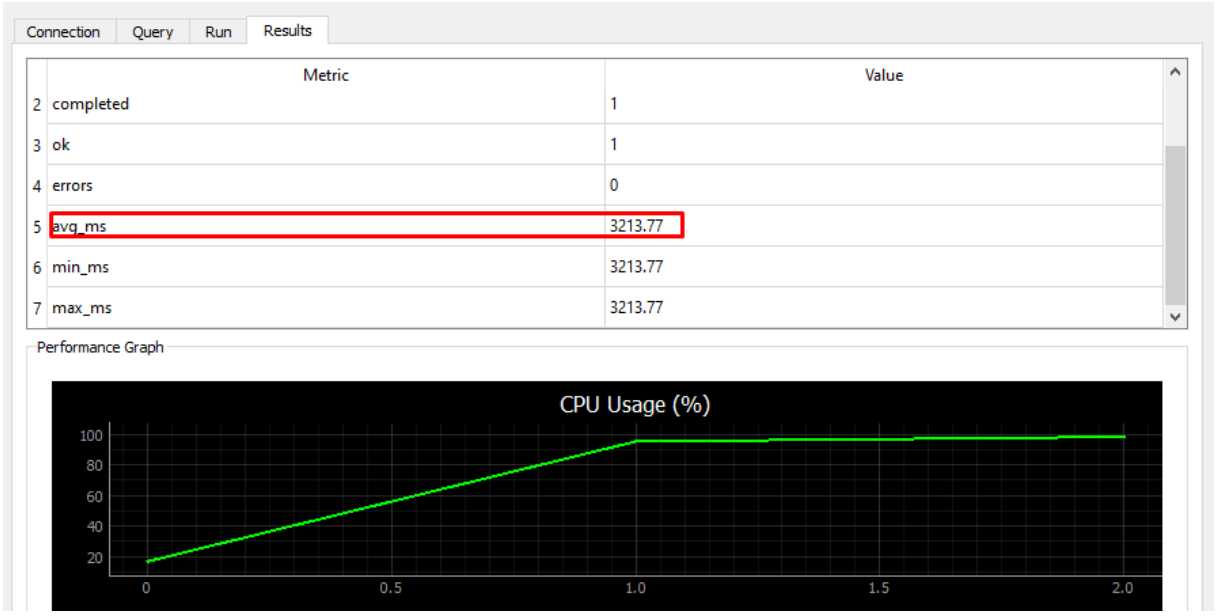


Slika 31. Rezultati testa preko vlastite aplikacije

5.4.3. Rezultat H2_Q3_ST3

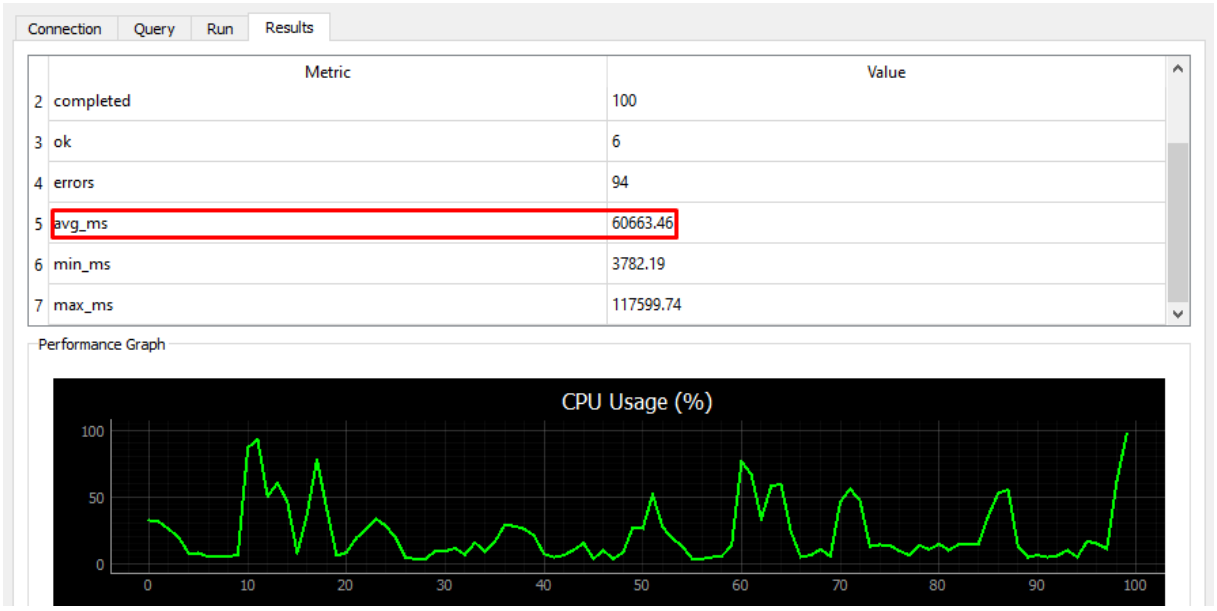
Rezultati bez summary tabele:

Prvi slučaj testiranja je kao i uvijek 1 korisnik 1 zahtjev. Za ovaj slučaj odziv je brz, za izvršenje upita je potrebno 3.2 sekunde, a CPU iskorištenost raste do 100%.



Slika 32. Rezultati testa preko vlastite aplikacije

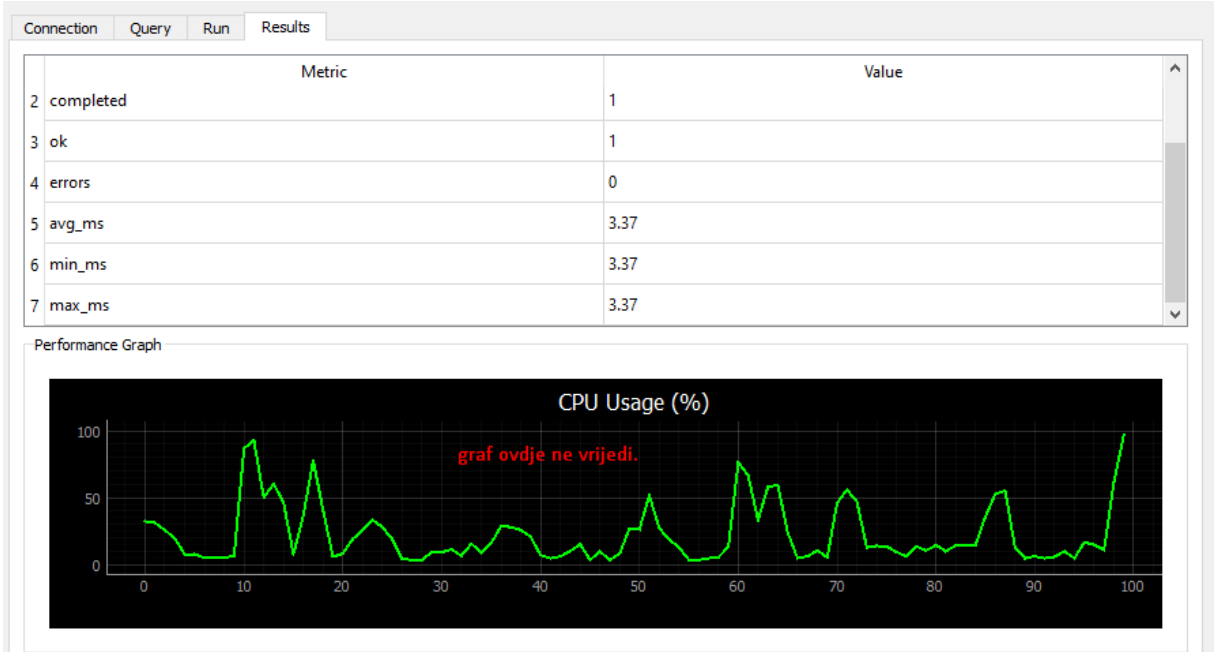
Naredno je testirano nad 50 korisnika koji šalju po 2 zahtjeva svakih 0.1s. Rezultat je sljedeći: prosječno vrijeme izvršavanja je 1 minuta, najmanje je 3s a najduže 117s, tj skoro 2 minute.



Slika 33. Rezultati testa preko vlastite aplikacije

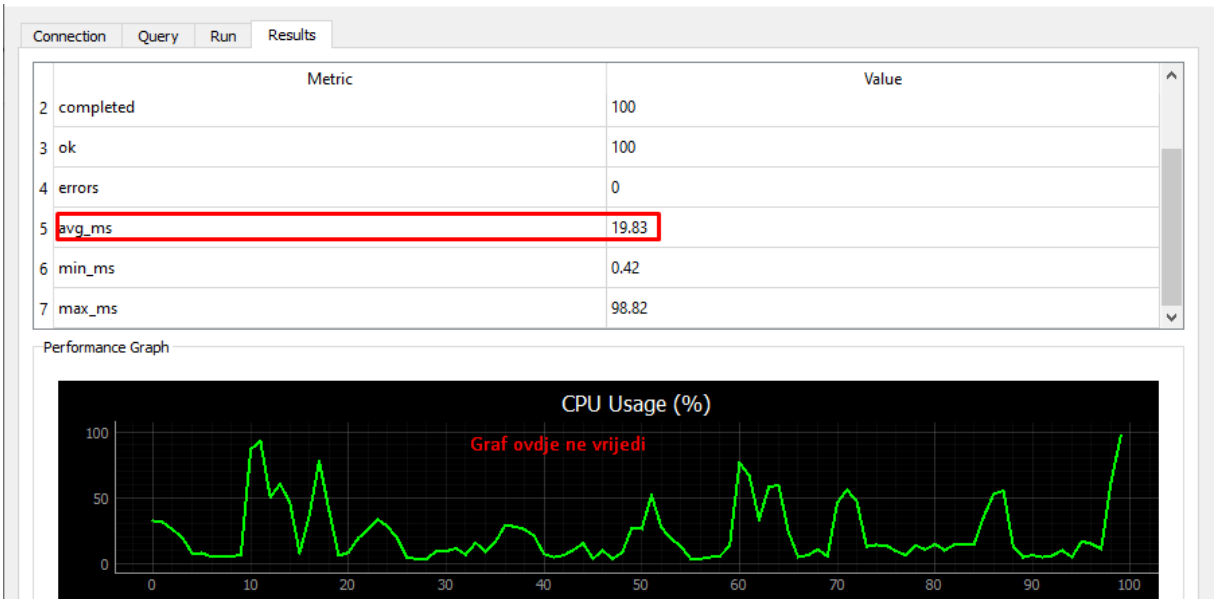
Rezultati sa summary tabelom:

Kao što je vidljivo na slici ispod, vrijeme izvršavanja za 1 korisnika i jedan zahtjev je 3.37ms, što je hiljadu puta brže nego bez summary tabele.



Slika 34. Rezultati testa preko vlastite aplikacije

Nakon dokazivanja sa jednim korisnikom, naredni test je 50 korisnika sa po 2 zahtjeva u razmaku od po 0.1s. Rezultati su zapanjujući. Prosječno vrijeme izvršavanja je 19ms, što je preko 3000 puta brže vrijeme izvršavanja nego bez summary tabele. Minimalno vrijeme izvršavanja je kraće ćak od 1ms i iznosi 0.43ms, a maksimalno vrijeme izvršavanja, tj. najduže čekanje je oko 0.1 sekunde. Ovo pokazuje visoku isplativost summary tabele nad update upitima.

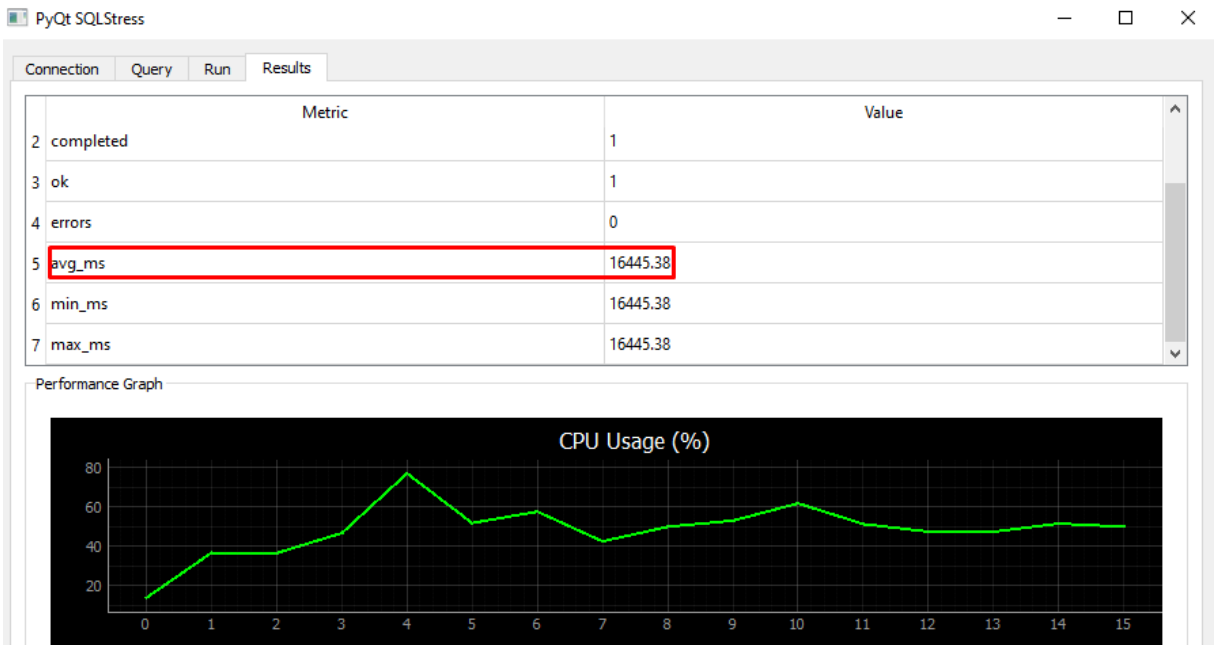


Slika 35. Rezultati testa preko vlastite aplikacije

5.4.4. Rezultat H2_Q4_ST4

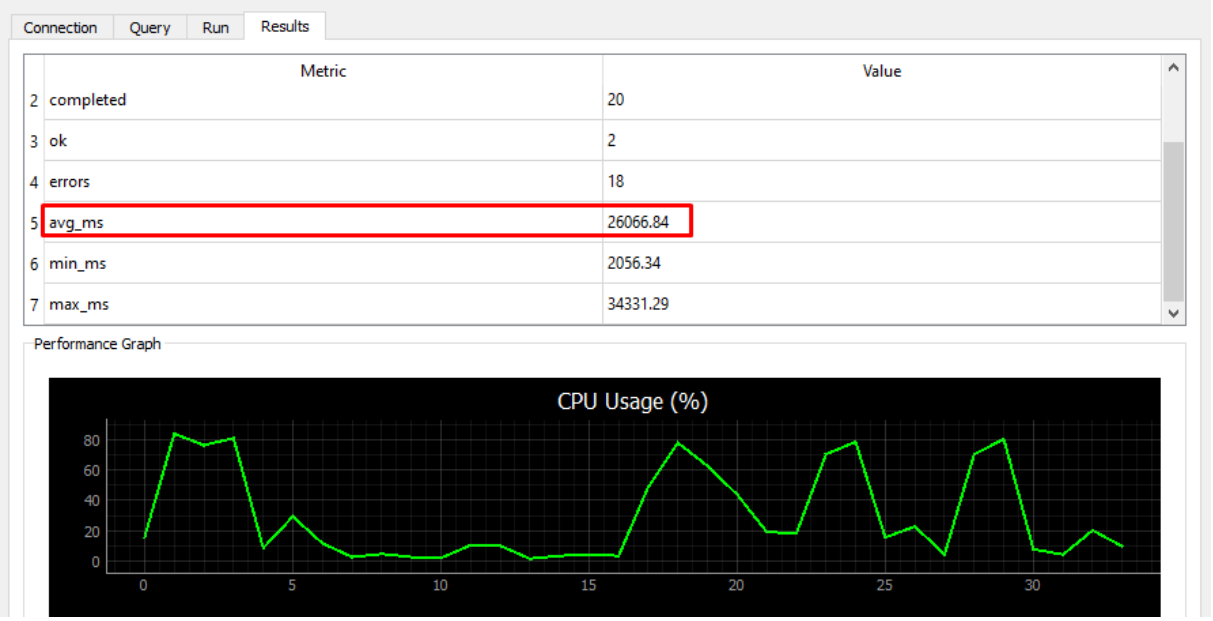
Rezultat testiranja bez summary tabele:

Kao i uvijek prvi test je izvršen nad 1 korisnikom i jednim zahtjevom. Na odgovor se čeka malo duže iz razloga što se prelazi čitava baza, to jeste čitava tabela Votes koja sadrži puno zapisa i traži se VoteType 5. Rezultat je sljedeći: čekanje na odziv traje oko 16s, a iskorištenost CPU raste do 80%.



Slika 36. Rezultati testa preko vlastite aplikacije

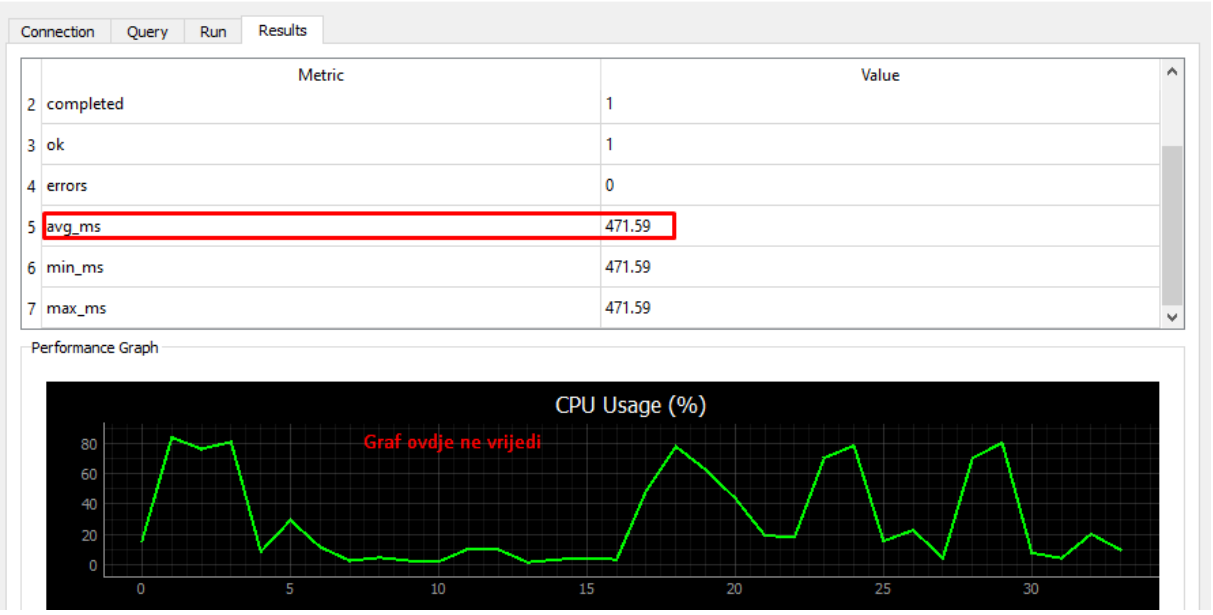
Naredno je testirano šta će se desiti kada bi 20 korisnika u isto vrijeme poslalo ovaj zahtjev (nije baš praktično da brišu svi istu bazu, ali ovo koristim u prezentacione svrhe). Prosječno trajanje odziva je oko 26s, tj tačno 26066ms. Vrijeme se kreće između 20s i 34s. Opterećenost CPU doseže 80%. Isti ovi testovi su kasnije ponovljeni sa summary tabelom.



Slika 37. Rezultati testa preko vlastite aplikacije

Rezultat testiranja sa summary tabelom:

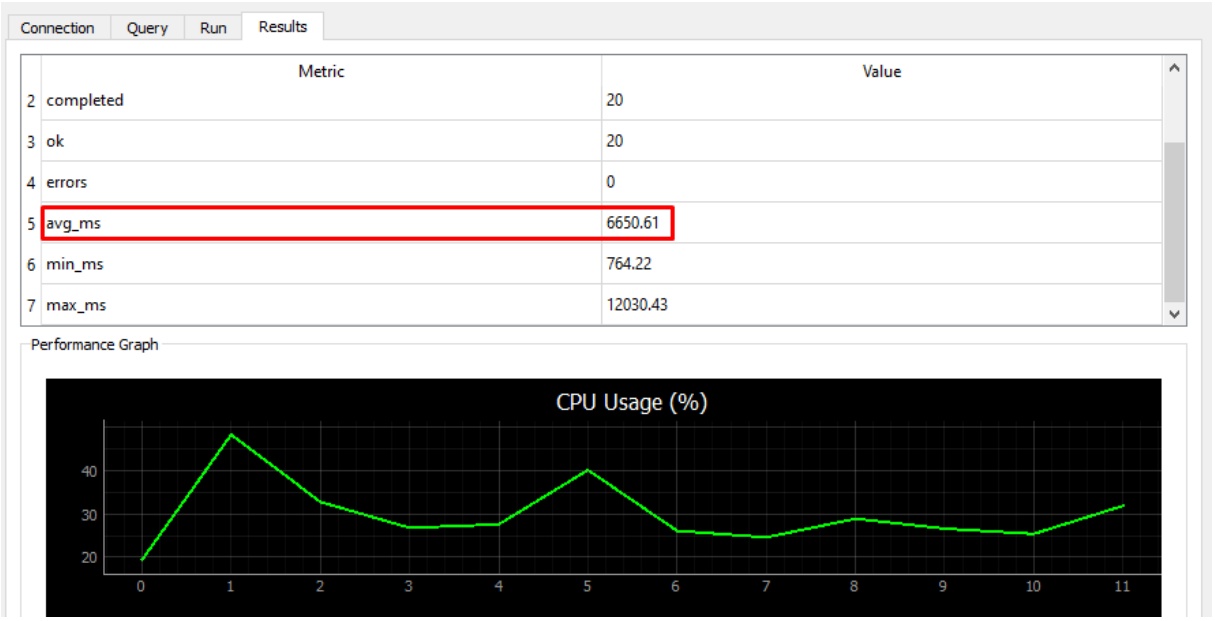
Kao i prethodno testirala sam 1 korisnika i jedan zahtjev. Virjeme odziva je bilo oko 0.5s, 32 puta brže nego bez summary tabele.



Slika 38. Rezultati testa preko vlastite aplikacije

Naredni test je bio nad 20 korisnika koji šalju ovaj zahtjev u isto vrijeme. Napomenula sam da ovo nike realističan scenario (da korisnici brišu upis u čitavoj bazi), ali koristim u prezentacione svrhe. Rezultati su puno bolji nego bez summary tabele. Prosječno vrijeme izvršavanja je 6650ms, što je oko 4 puta brže nego bez summary tabele. Najkraće vrijeme je ispod 1 sekunde a najduže vrijeme je 12s, dok je najduže vrijeme bez summary tabele 34

sekunde. Opterećenost CPU doseže 40%, što je duplo manje nego bez summary tabele. Dakle- što ose tiče resursa, ovo je puno bolji 'prilaz' čitavoj bazi. Dakle i ovaj test je potvrdio moju hipotezu.



Slika 39. Rezultati testa preko vlastite aplikacije

5.5. Konačni zaključak za Hipotezu 2

Hipoteza 2 se potvrdila.

Denormalizovane summary tabele značajno su poboljšale performanse agregacijskih upita. Najveće ubrzanje ostvareno je kod SELECT i UPDATE upita, dok su i kod DELETE operacija pokazale jasnu prednost. Kod INSERT upita razlike su bile manje izražene. Dakle, hipoteza se u cjelini smatra **potvrđenom.**

Nakon svega, pružam kratki osvrt na rezultat svakog testiranja:

- H2_Q1_ST1- Select upit. Testiranje prvog upita pokazalo je značajnu razliku u performansama između rada bez summary tabele i sa summary tabelom. Kada se upit izvršavao direktno nad tabelom Votes, prosječno vrijeme odgovora iznosilo je oko 3.8 sekundi pri jednom klijentu i jednom zahtjevu. Međutim, pri korištenju summary tabele, isti rezultat je dobijen za svega 5 milisekundi, što je više od 700 puta brže. I pri testu sa 100 klijenata koji šalju zahtjeve u kratkom razmaku, prosječno vrijeme odgovora ostalo je zanemarivo (oko 32 ms), dok je kod rada bez summary tabele prosjek bio preko 4 minute. Ovo jasno potvrđuje da summary tabele značajno poboljšavaju performanse agregacijskih upita i opravdavaju postavljenu hipotezu.

H2_Q1_ST1					
Broj korisnika	Broj iteracija	Q (t + CPU)		ST (t + CPU)	
1	1	3790 ms	100%	5ms	100%
100	1 /0.1s	280292 ms	100%	32ms	100%

Tabela 5. Rezultati testiranja H2_Q1_ST1

- H2_Q2_ST2- Insert upit. Rezultati testa pokazali su da kod jednostavnih INSERT operacija nema značajne razlike u performansama između rada sa i bez summary tabele. Za jednog korisnika vrijeme izvršavanja oba pristupa bilo je praktično zanemarljivo (7.4 ms bez summary tabele, 1.3 ms sa summary tabelom). Tek pri opterećenju sa 100 korisnika u isto vrijeme došlo je do uočljivije razlike: prosjek bez summary tabele iznosio je 3.4 ms, dok je sa summary tabelom bio 120 ms. Ipak, oba vremena su dovoljno kratka da se može reći da gotovo nema razlike između korištenja summary tabele i nekorištenja iste.

H2_Q2_ST2					
Broj korisnika	Broj iteracija	Q (t + CPU)		ST (t + CPU)	
1	1	7.43 ms	100%	1.3 ms	100%
100	1	3.39 ms	100%	120 ms	100%

Tabela 6. Rezultati testiranja H2_Q2_ST2

- H2_Q3_ST3- Update upit.Rezultati testa 3 pokazuju ogroman kontrast između rada bez i sa summary tabele. Kada se UPDATE izvršavao direktno nad tabelom Votes, za jednog korisnika i jedan zahtjev prosječno vrijeme trajanja bilo je oko 3.2 sekunde, a CPU je dostizao i do 100% opterećenja. Pri većem opterećenju, sa 50 korisnika koji šalju po dva zahtjeva u kratkom intervalu, prosječno vrijeme izvršavanja dostizalo je i više od jedne minute, uz izrazito nestabilne vrijednosti između minimalnog i maksimalnog trajanja.

  Nasuprot tome, korištenjem summary tabele prosječno trajanje ažuriranja za jednog korisnika iznosilo je svega 3.37 milisekundi, što je preko hiljadu puta brže. Kada je test ponovljen sa 50 korisnika, prosječno vrijeme trajanja bilo je samo 19.8 ms, a maksimalno čekanje je ostalo ispod 0.1 sekunde. Ovo jasno potvrđuje da denormalizovane summary tabele imaju ogromnu prednost kod ažuriranja podataka u scenarijima agregacije, čime se u potpunosti opravdava hipoteza.

H2_Q3_ST3					
Broj korisnika	Broj iteracija	Q (t + CPU)		ST (t + CPU)	
1	1	3213 ms	100%	3.37 ms	100%
50	2 /0.1s	60663 ms	100%	19.83 ms	100%

Tabela 7. Rezultati testiranja H2_Q3_ST3

- H2_Q4_ST4- Delete upit. Rezultati testa brisanja Favorite glasova pokazali su jasnu razliku u performansama između rada sa i bez summary tabele. Kada je brisanje vršeno direktno nad tabelom Votes, vrijeme izvršavanja za jednog korisnika iznosilo je oko 16 sekundi, dok je sa 20 korisnika prosjek trajao više od 26 sekundi, uz konstantno visoko opterećenje CPU-a (oko 80%). Sa druge strane, korištenjem summary tabele vrijeme odziva je bilo svega 0.5 sekundi za jednog korisnika, odnosno prosječno 6.6 sekundi za 20 korisnika, što je višestruko brže. Također, opterećenje CPU-a bilo je upola manje (40%). Ovaj test potvrđuje da summary tabele značajno poboljšavaju performanse i kod DELETE operacija.

H2_Q4_ST4					
Broj korisnika	Broj iteracija	Q (t + CPU)		ST (t + CPU)	
1	1	16445ms	80%	471 ms	80%
20	1	26066 ms	80%	6650 ms	40%

Tabela 8. Rezultati testiranja H2_Q4_ST4

6. HIPOTEZA 3

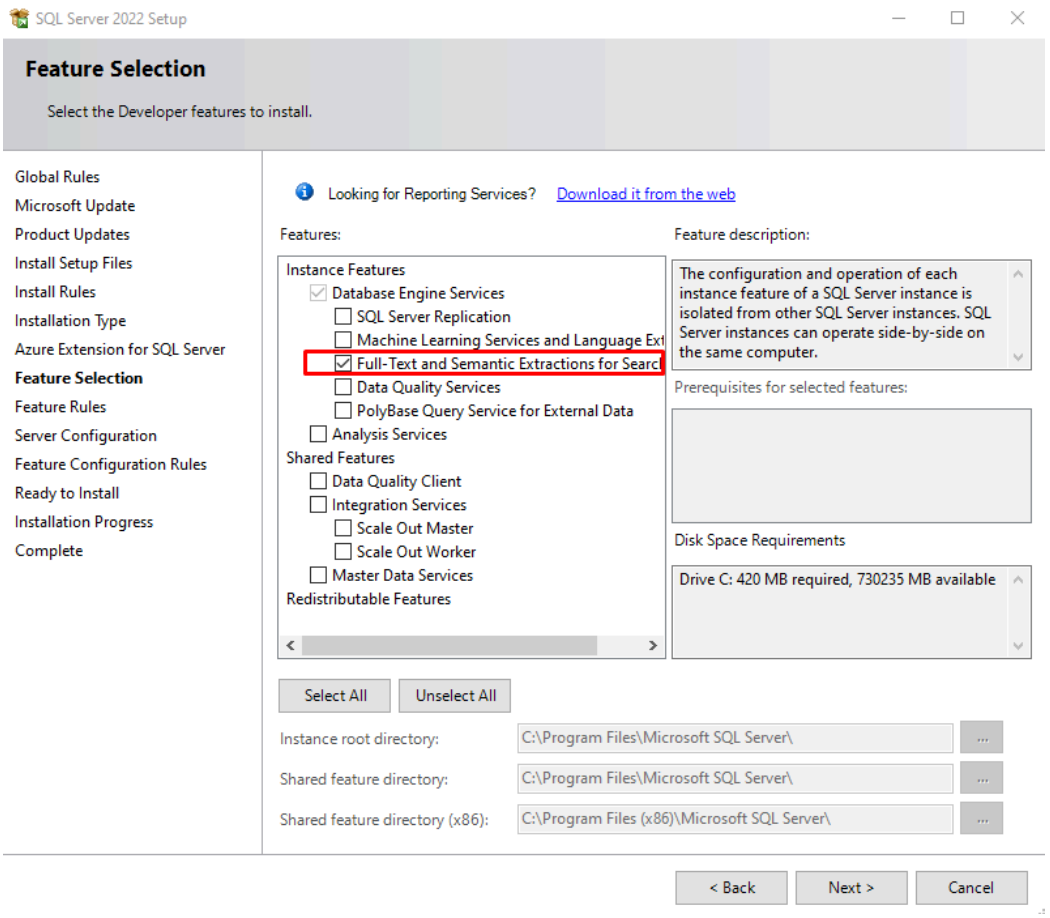
Full-Text Search (FTS) značajno smanjuje vrijeme pretrage velikih tekstualnih polja u odnosu na LIKE.

Upiti koji koriste LIKE na velikim tekstualnim poljima zahtijevaju čitanje i skeniranje svih znakova, što je sporo. FTS koristi indeksiranje riječi, što omogućava bržu i tačniju pretragu. Ova metoda je naročito korisna kada korisnik u aplikaciji traži određene fraze unutar sadržaja i sistem mora brzo odgovoriti bez opterećivanja servera.

6.1. Osnovni pojmovi

- *LIKE operator*- SQL operator koji se koristi za pretragu tekstualnih vrijednosti pomoću obrazaca (pattern matching). Radi tako što prolazi kroz svaku vrijednost u koloni i poredi znak po znak. Kod malih tabela ovo ne predstavlja problem, ali kod velikih tekstualnih polja dovodi do sporih performansi jer SQL Server mora da izvrši full table scan.
- *Full-Text Search (FTS)*- napredna tehnika pretrage u SQL Serveru koja omogućava indeksiranje riječi unutar tekstualnih polja. Umjesto linearnog prolaska kroz svaku vrijednost, FTS koristi specijalne full-text indekse koji ubrzavaju pronalaženje pojmova i fraza. Podržava pretragu po riječima, frazama, prefiksima i složenijim logičkim operatorima.
- *Full-Text Index*- posebna vrsta indeksa koja se kreira nad kolonom (npr. Title ili Body u tabeli Posts). Sadrži mapirane riječi i njihove pozicije unutar tekstualnog sadržaja, što omogućava da se pretraga izvrši u milisekundama umjesto u sekundama.
- *CONTAINS i FREETEXT* - funkcije koje se koriste uz FTS. CONTAINS omogućava pretragu preciznih riječi ili fraza, dok FREETEXT traži pojmove koji su slični ili imaju isto značenje kao upit korisnika.

Za potrebe testiranja napravljen je jedan Full-Text Index (npr. nad kolonom Body u tabeli Posts). To je “priprema terena”. Prvo je potrebno instalirati Full Text Search, modifikacijom već postojeće instance SQL Servera.



Slika 40. Način dodavanja Full Text Search funkcionalnosti

Ispod se nalazi upit koji kreira Full Text nad 2 kolone- Title i Body. Title je jednostavniji jer su kraći testovi, ali Body sadrži velike tekstove i tu bi FTS stvarno došao do izražaja.

```
CREATE FULLTEXT INDEX ON Posts
(
    Title LANGUAGE 1033,
    Body LANGUAGE 1033
)
KEY INDEX PK_Posts_Id
ON StackOverflowFTCatalog
WITH CHANGE_TRACKING AUTO;
```

CHANGE_TRACKING AUTO znači da će se indeks automatski ažurirati nakon INSERT/UPDATE/DELETE.

6.2. Način mjerenja performansi

Za uspoređivanje performansi koristi se:

- PyQT SQLStress (uvijek)- Vlastita aplikacija koja je klon SQLQueryStress ali ima dodatne funkcionalnosti i urađena je drugim programskim jezikom.

6.3. Testiranje hipoteze

6.3.1. Test H3_Q1_FTS1

Upit prikazan ispod ne koristi Full Text Search, a pretrazuje citavu tabelu Posts, upise u koloni ‘Body’. To jeste, traži tekstove objava koji sadrže riječi ‘performance’ i ‘query’,koji su nastali prije 2010 godine (prethodno provjereno u SSMS, ovi zapisi postoje).

```
SELECT TOP 50 Id, Title, Score, CreationDate
FROM Posts
WHERE Body LIKE '%performance%'
  AND Body LIKE '%query%'
  AND YEAR(CreationDate) >= 2010
ORDER BY Score DESC;
```

Naredni je upit pomoću Full text search-a:

```
SELECT TOP 50 Id, Title, Score, CreationDate
FROM Posts
WHERE CONTAINS(Body, 'performance AND query')
  AND YEAR(CreationDate) >= 2010
ORDER BY Score DESC;
```

6.3.2. Test H3_Q2_FTS2

Upit prikazan ispod ne koristi Full text Search. Razlog zašto u testu 2 radim odmah sa UPDATE , mjesto kao inače sa INSERT jeste taj što se pokazalo da INSERT nema logike testirati za ovu hipotezu, smaim time ga i preskačem.

Želim da se svugdje u tekstu, gdje postoji ‘query’ upiše ‘SQL query’.

```
UPDATE Posts
SET Body = REPLACE(Body, 'query', 'SQL query')
WHERE Body LIKE '%query%';
```

Ispod se nalazi ista naredba ali odrađena sa Full Text Search:

```
UPDATE Posts
SET Body = REPLACE(Body, 'query', 'SQL query')
WHERE CONTAINS(Body, 'query');
```

6.3.3. Test H3_Q3_FTS3

Finalni test je rađen sa naredbom DELETE. Iz baze želim da obrišem sve objave koje sadrže riječ ‘query’ narednim upitom bez FTS:

```
DELETE FROM Posts
WHERE Body LIKE '%query%';
```

Naredni upit radi istu stvar ali korištenjem FTS.

```
DELETE FROM Posts
WHERE CONTAINS(Body, 'query');
```

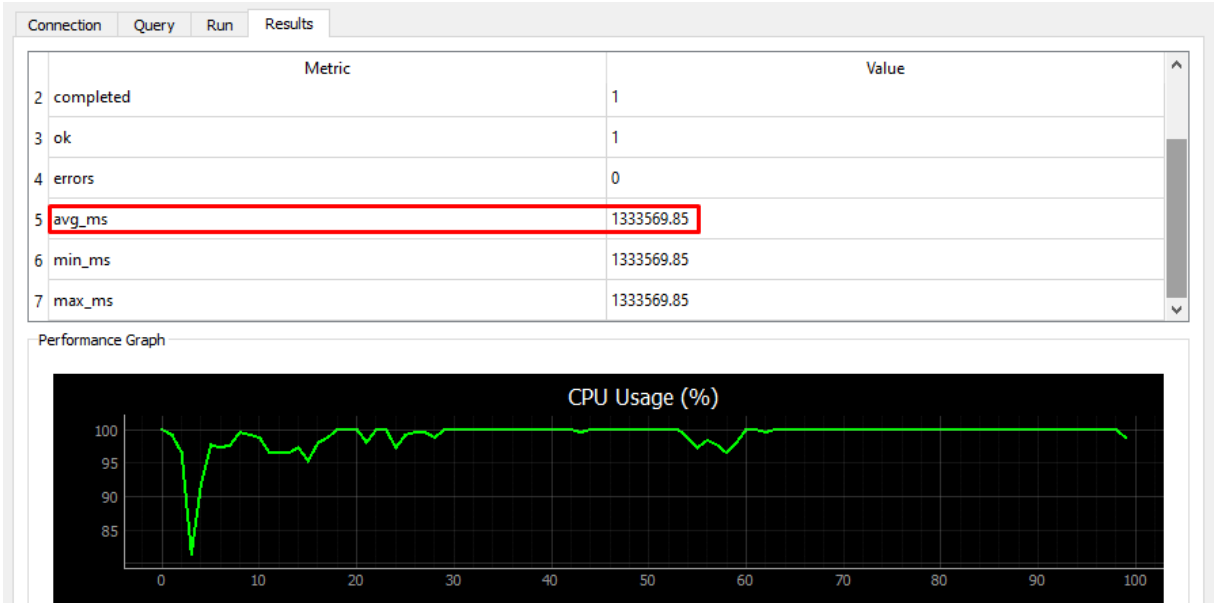
Upiti nisu kompleksni sami po sebi,gledajući isntaksno, ali zahtjevaju mnogo rada unutar same baze podataka, pa su zato odabrani kao testni primjeri.

6.4. Rezultati testiranja

6.4.1. Rezultat H3_Q1_FTS1

Rezultat bez Full text search:

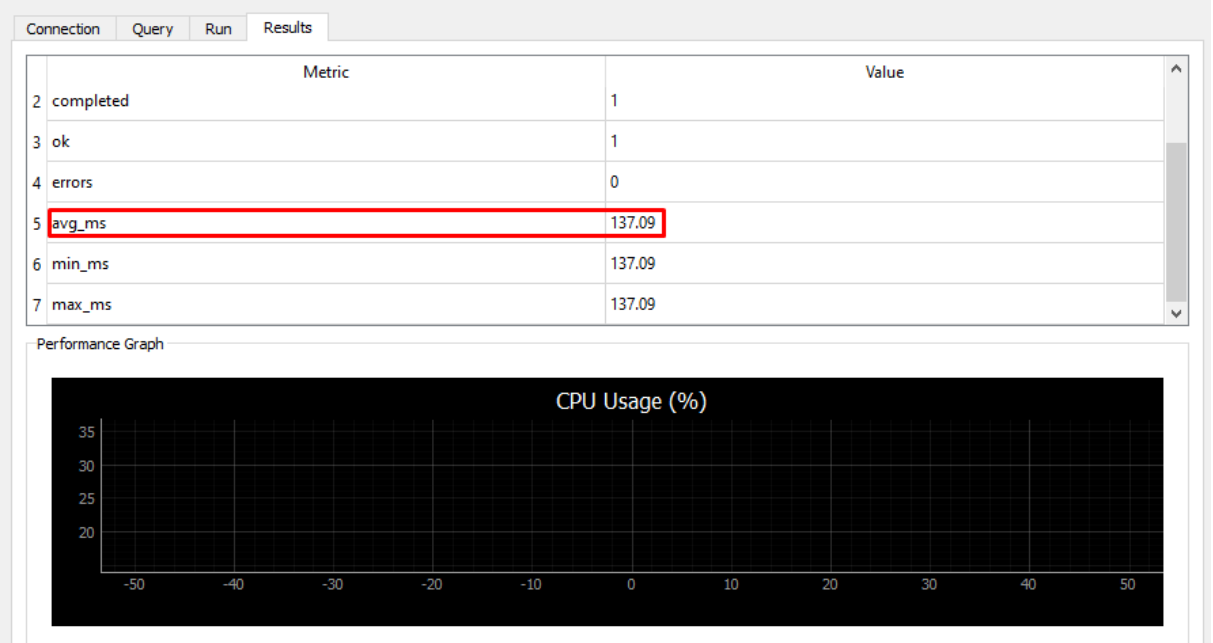
Pokretanje upita bez Full text search je zahtjevalo mnogo vrmena. Prvo, kao i svaki put je pokrenut nad samo jednim korisnikom i jednim zahtjevom, na slici ispod su vidljivi rezultati iz moje aplikacije. Iskorištenost CPU se kretala između 85 i 100%. Vrijeme odziva je bilo (dosad najduže) 1333569ms, što je oko 1335s ili preko 22 minute. Za ovako ‘jednostavan’ upit, ovo je baš dugo vrijeme izvršavanja. Naredni test bi trebao biti nad više korisnika, ali je očigledno da će iziskivati još više vremena, mogu slobodno prepostaviti da možda i do sad vremena, pa nema potrebe za takvim testom i ovo je dovoljna informacija.



Slika 41. Rezultati testa preko vlastite aplikacije

Rezultat sa Full text search:

Nakon pokretanja testa bez FTS, pokrenut je jedan sa, isto nad 1 korisnikom i jednim zahtjevom. Rezultati su drastično bolji. Rezltat direktno iz aplikacije je prikazan na slici ispod. Za odgovor je trebalo oko 130ms, dakle oko 0.1s. Ovo vrijeme je neznatno, a pogotovo puno bolje od 22 minute koliko je trebalo bez Full text search-a. Iskreno rečeno, bila sam nsigurna u ove rezultate i isti upit odlučila pokrenuti direktno u SSMS, da vidim da li će se vremena podudarati.



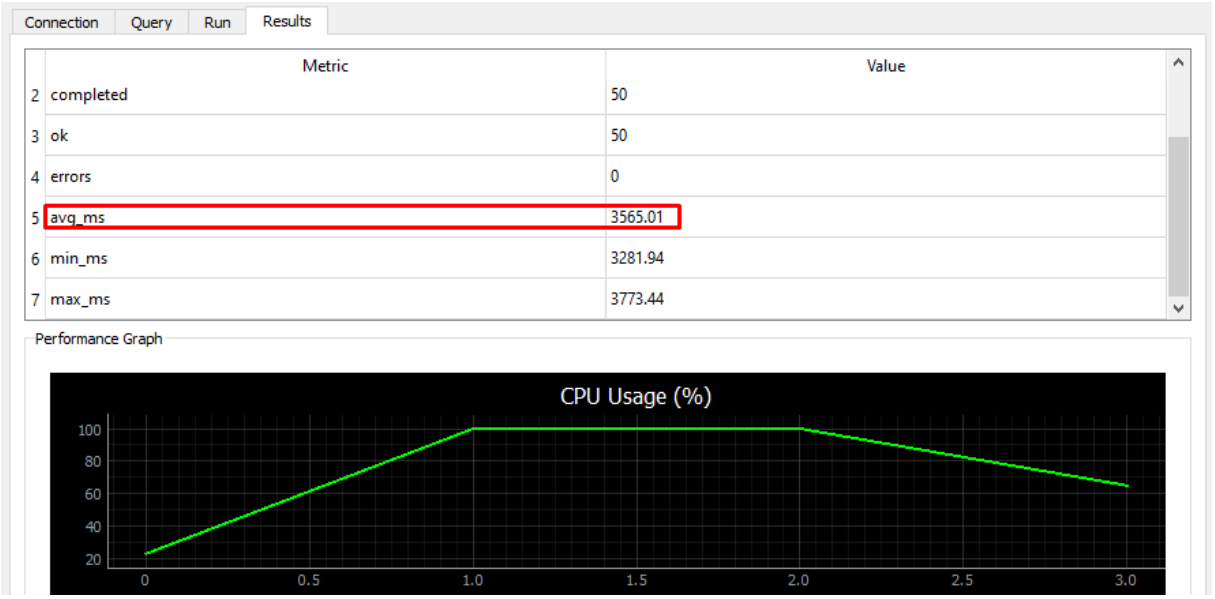
Slika 42. Rezultati testa preko vlastite aplikacije

Vrijeme izvršavanja unutar SSMS je bilo 156ms. Što potvrđuje rezultat iz moje aplikacije. Client statistics ovog testa putem SSMS su vidljivi na slici ispod:

	Trial 1	Average
Client Execution Time	13:08:08	
Query Profile Statistics		
Number of INSERT, DELETE and UPDATE statements	0 →	0.0000
Rows affected by INSERT, DELETE, or UPDATE stateme...	0 →	0.0000
Number of SELECT statements	3 →	3.0000
Rows returned by SELECT statements	58 →	58.0000
Number of transactions	0 →	0.0000
Network Statistics		
Number of server roundtrips	3 →	3.0000
TDS packets sent from client	3 →	3.0000
TDS packets received from server	8 →	8.0000
Bytes sent from client	604 →	604.0000
Bytes received from server	41597 →	41597.0000
Time Statistics		
Client processing time	4 →	4.0000
Total execution time	156 →	156.0000
Wait time on server replies	152 →	152.0000

Slika 43. Client statistics za ovaj upit

Radi brzine izvrženja ovog upita, pokušala sam testirati nad 50 klijenata koji šalju isti zahtjev u razmaku od po 0.1s. Za izvršenje je u prosjeku bilo potrebno 3.5 sekunde. Dakle- korištenjem običnog pretraživanja i poređenja sa LIKE , jedan klijent bi čekao na odgovor preko 20 minuta, dok bi pretraživanjem nakon formiranog FTS, čak 50 klijenata koji šalju zahtjeve u vrlo kratkom roku, moglo dobiti odgovor u ispod 3 sekunde. Ovime je hipoteza 3 dokazana u smislu SELECT upita.

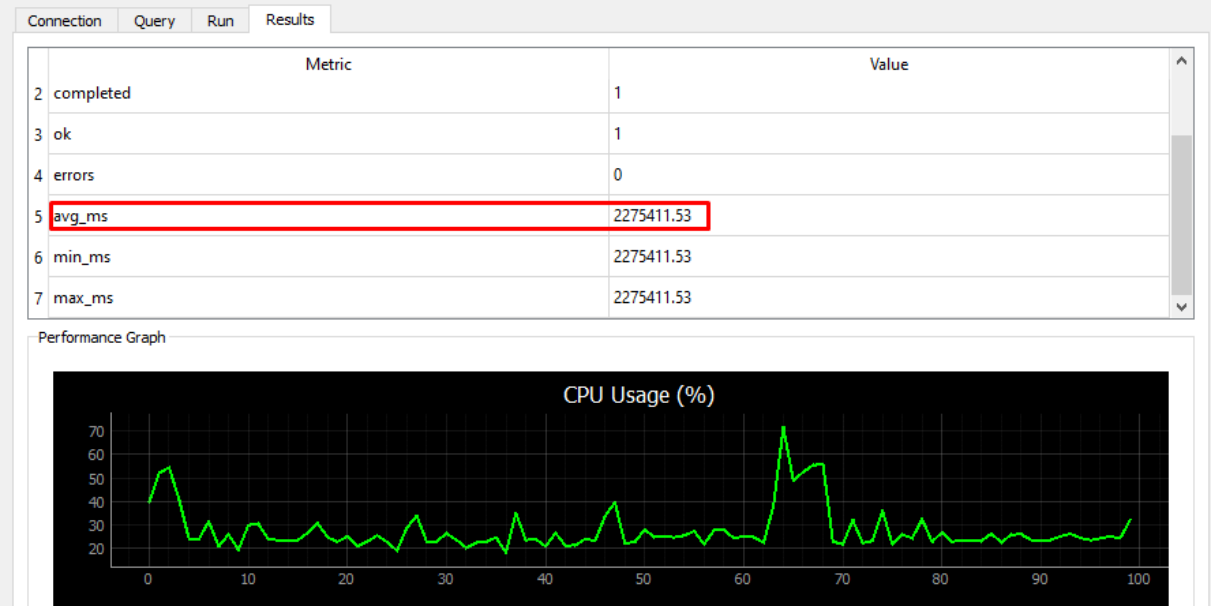


Slika 44. Rezultati testa preko vlastite aplikacije

6.4.2. Rezultat H3_Q2_FTS2

Rezultat bez Full Text Serach:

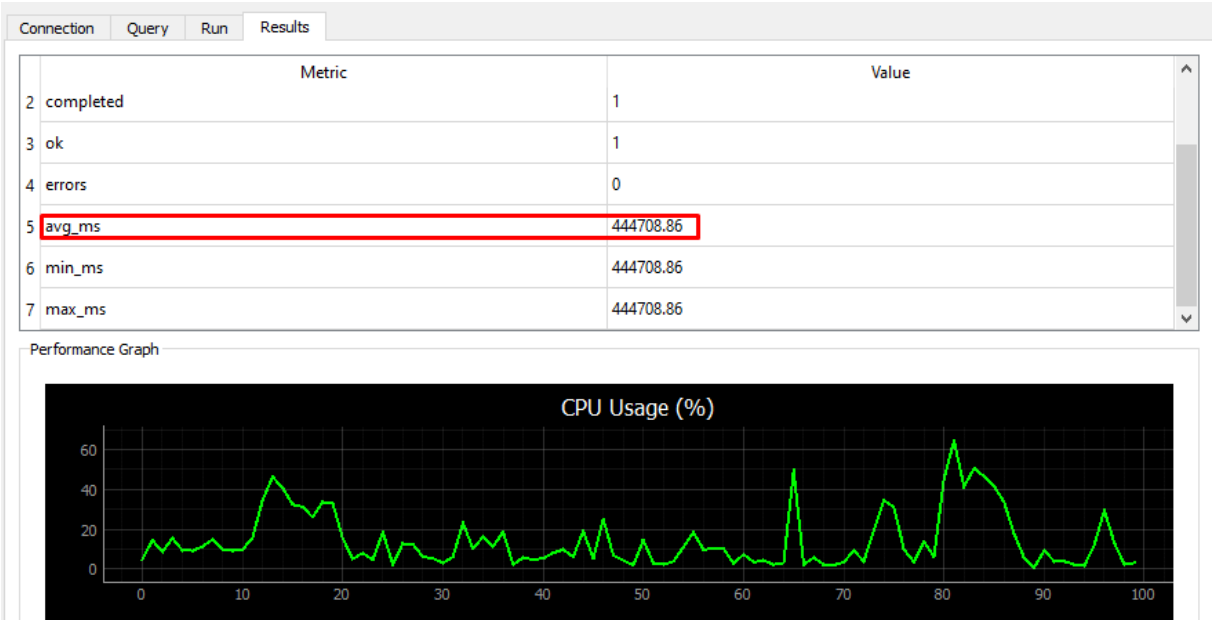
Bez korištenja FTS, rezultati su opet užasni. Na test samo 1 korisnik i jedan zahtjev vrijeme odziva je trajalo 2275411 ms, to jeste oko 37 minuta je bilo potrebno čekati na odgovor. S obzirom na ovoliko vrijeme čekanja, nema potrebe testirati na više korisnika i možemo slobodno pretpostaviti da bi vrijeme odziva u slučaju više korisnika prelazilo u sate i ne bi više bilo u minutema.Korištenje CPU raste do 70%.



Slika45. Rezultati testa preko vlastite aplikacije

Rezultat sa Full text Search:

Sa FTS rezultati su bili bolji i vidljivi su na slici ispod. Test je 1 korisnik 1 zahtjev. Vrijeme odziva je bilo 444708 ms ili oko 7 minuta. To je i dalje dugo vremena potrebnog za odziv, ali je definitivno bolje nego 34 minute koliko je trebalo bez FTS. Dakle vrijeem izvršavanja je 5 puta brže nego bez FTS. Također bitno je napomenuti da ovdje opterećenje CPU dostiže samo 60%, dok je prethodno dostizalo i do 80%.

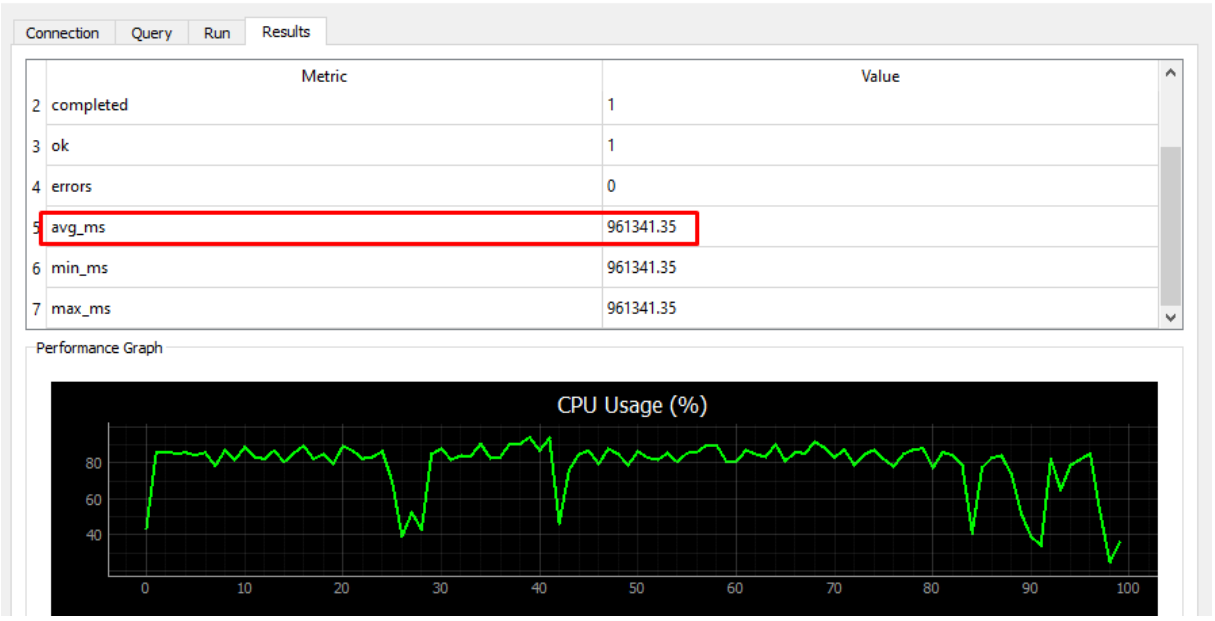


Slika 46. Rezultati testa preko vlastite aplikacije

6.4.3. Rezultat H3_Q3_FTS3

Rezultati bez Full Text Search:

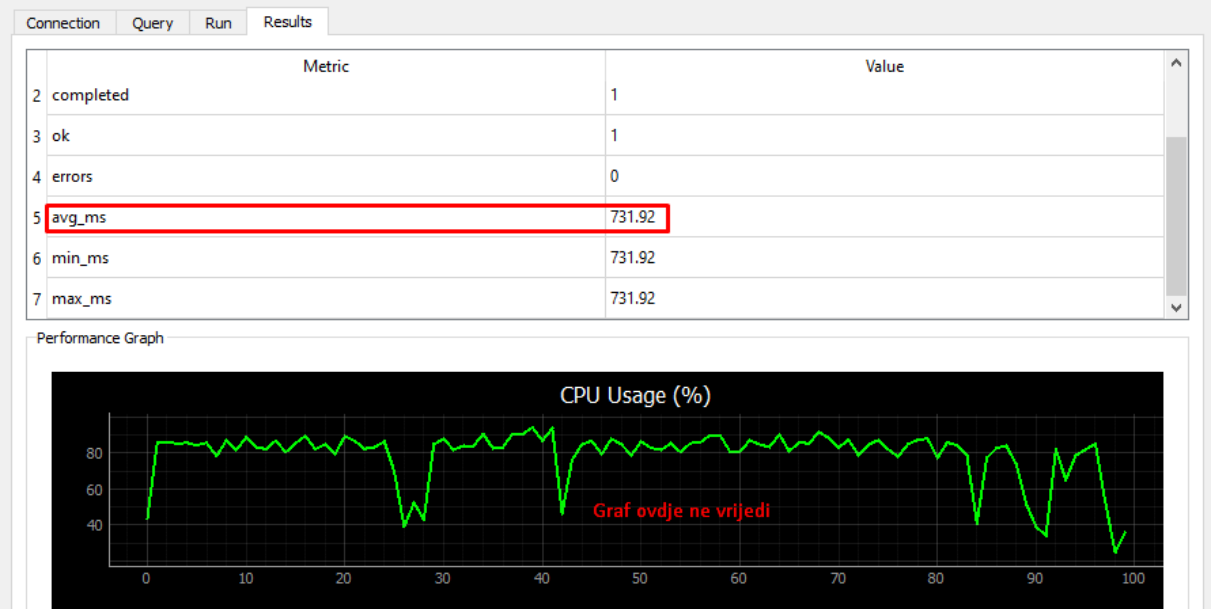
Prosječno vrijeme odziva za jednog klijenta i jedan zahtjev bez FTS je bilo čak 961341ms , što je malo više od 16 minuta. Test nike realističan primjer, jerneće se desiti da jedan klijent briše sve postove, ali zamislimo da briše velik broj svojih postova, u hiljadama.. ovo bi zahtjevalo neko proporcionalno vrijeme. Korištenjee resursa CPU je dosezalo oko 80% maksimalno.



Slika 47. Rezultati testa preko vlastite aplikacije

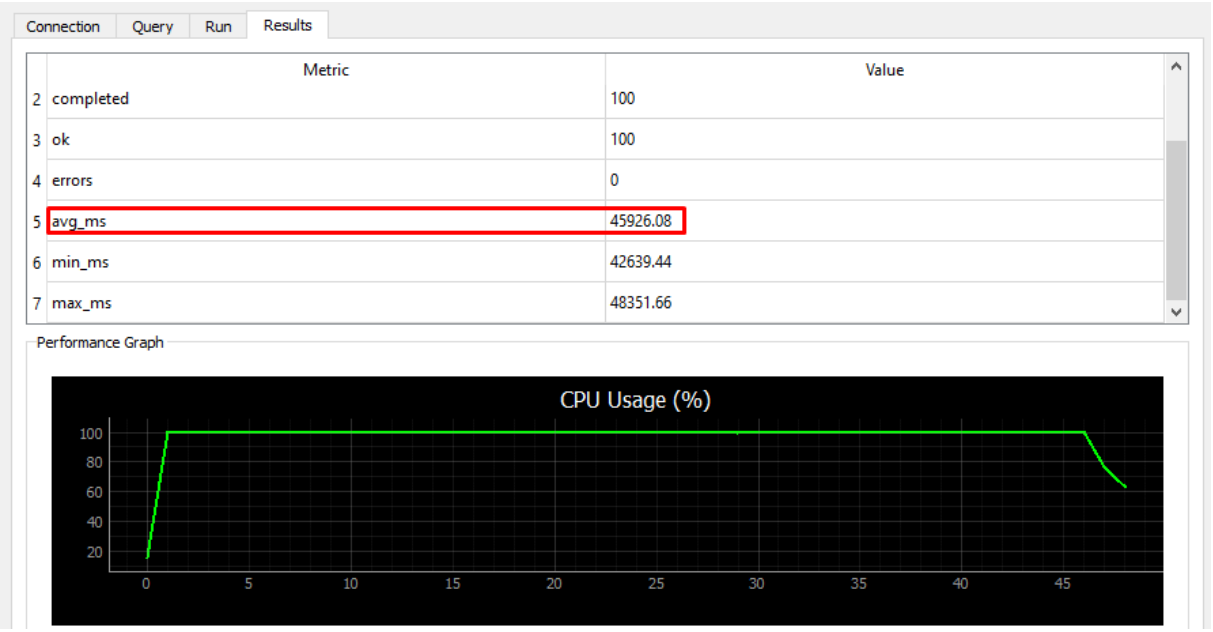
Rezultati sa Full Text Search:

Nakon tests bez FTS ista stvar je testirana nad 1 klijent i 1 zahtjev. Rezultat je puno bolji. Vrijeme odziva je 731ms, dakle manje od 1s. Što je oko 1700 puta brže nego bez FTS. Iz razloga ovako dobrih performansi, testirano je i na više klijenata i više zahtjeva.



Slika 48. Rezultati testa preko vlastite aplikacije

Naredno sam testirala nad 100 klijenata koji šalju zahtjeve za delete u razmaku od 0.1s. Rezultat je i dalje puno bolji nego nego bez FTS i to samo za jednog klijenta. Za odziv je u prosjeku trebalo 45 sekundi (20ak puta brže nego za jednog klijenta bez FTS). Najkraći odziv je trajao 42 sekunde, a najduži odgovor 48 sekundi.



Slika 49. Rezultati testa preko vlastite aplikacije

6.5. Konačni zaključak za Hipotezu 3

Hipoteza 3 se potvrdila.

Full-Text Search značajno je unaprijedio performanse tekstualnih upita u poređenju s klasičnom pretragom pomoću LIKE operatora. Najveće ubrzanje ostvareno je kod SELECT upita, gdje je FTS pokazao i do 10.000 puta bolje rezultate. Kod UPDATE upita ubrzanje je bilo nešto umjerenije, ali i dalje višestruko u korist FTS-a. Kod DELETE operacija također je zabilježena drastična razlika u korist FTS-a. Dakle, hipoteza se u cjelini smatra **potvrđenom**.

Dalje pružam kratak pregled rezultata testiranja Hipoteze 3:

- H3_Q1_FTS1- Select upit. Rezultati su pokazali ogromnu razliku u performansama između klasične pretrage pomoću LIKE i Full-Text Search (FTS). Kod upita sa LIKE, izvršavanje je trajalo preko 22 minute uz maksimalno opterećenje CPU-a (85-100%). Nasuprot tome, isti upit sa FTS (CONTAINS) vratio je rezultat za svega 130-150 milisekundi, što je više oko 10.000 puta brže.

  Čak i u scenariju sa 50 klijenata koji istovremeno šalju zahtjev, FTS je pružio odgovor u prosjeku za 3,5 sekunde, dok bi obična pretraga bila potpuno nepraktična. Ovaj test jasno pokazuje da FTS drastično smanjuje vrijeme pretrage velikih tekstualnih polja i potvrđuje postavljenu hipotezu

H3_Q1_FTS1					
Broj korisnika	Broj iteracija	Q (t + CPU)		FTS (t + CPU)	
1	1	1333569 ms	100%	137ms	40%
50	1			3565ms	100%

Tabela 9. Rezultati testiranja H3_Q1_FTS1

- H3_Q2_FTS2- Update upit. Rezultati testa pokazali su da Full-Text Search donosi značajno bolje performanse i kod ažuriranja podataka. Bez FTS-a, prosječno vrijeme izvršavanja upita iznosilo je oko 37 minuta, uz visoko opterećenje CPU-a do 80%.

  Sa FTS-om, isto ažuriranje završeno je za oko 7 minuta, što je približno 5 puta brže, a opterećenje je dosezalo 60%. Iako je i dalje riječ o dugom trajanju, razlika jasno pokazuje prednost korištenja FTS nad klasičnim LIKE pretragama

H3_Q2_FTS2					
Broj korisnika	Broj iteracija	Q (t + CPU)		FTS (t + CPU)	
1	1	2275411 ms	80%	444708ms	60%

Tabela 10. Rezultati testiranja H3_Q2_FTS2

- H3_Q3_FTS3- Delete upit. Bez korištenja Full-Text Search (FTS), prosječno vrijeme odziva za jednog klijenta i jedan zahtjev iznosilo je preko 16 minuta (961.341 ms), uz opterećenje CPU-a i do 80%. Nakon primjene FTS-a, isti upit je završen za 731 ms,

što predstavlja poboljšanje od približno 1700 puta. Dodatno, kod testiranja sa 100 klijenata koji šalju zahtjeve u kratkom razmaku, prosječno vrijeme odziva bilo je oko 45 sekundi, što je i dalje višestruko brže od rada bez FTS-a.

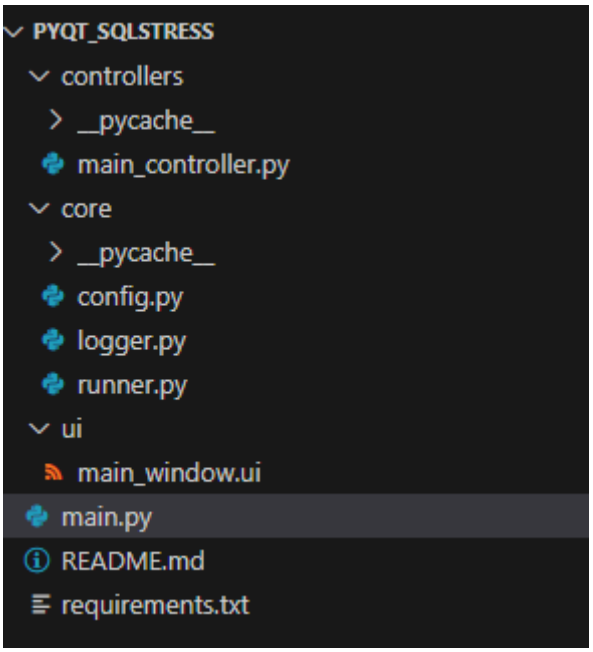
H3_Q3_FTS3					
Broj korisnika	Broj iteracija	Q (t + CPU)		FTS (t + CPU)	
1	1	961341 ms	80%	731ms	80%
100	1/ 0.1S			45928ms	100%

Tabela 11. Rezultati testiranja H3_Q2_FTS2

7. IZRADA I PREGLED APLIKACIJE

Kao što sam već kroz početak rada pomenula- za testiranje izvršavanja par upita je korišten SQLStressQuery. Ova aplikacija je opensource aplikacija izrađena u programskom jeziku C# te služi za pregled vremena izvršavanja i performansi određenih upita. Ista ova aplikacija ima mogućnost simuliranja više klijenta i perioda u kojem šalju upite.
Nakon korištenja ove aplikacije došla sam na ideju da je vrlo lahko napraviti klon, koristeći PyQt, tj. Python programski jezik. PyQt sam izabrala radi jednostavnosti pakovanja čitave aplikacije u jedan .exe fajl, koristeći biblioteku PyInstaller, te naravno radi prethodnog iskustva sa radom u izradi ovakvih malih aplikacija.

Kroz ovo poglavlje ću uraditi kratku analizu koraka izrade aplikacije, zatim pregled same aplikacije i ukazati na eventualne izazove sa kojima sam se susrela.



Slika 50. Struktura folders aplikacije

Aplikacija je klasična PyQT aplikacija, sa lijepo razvrstanim fajlovima, gdje svaki od njih igra svoju ulogu:

- **core**- za konfiguraciju, logiku izvršavanja i konekciju,
- **controllers**- za logiku i povezivanje dugmadi,
- **ui**- za interfejs.

Core folder sadrži 3 fajla: config, logger i runner:

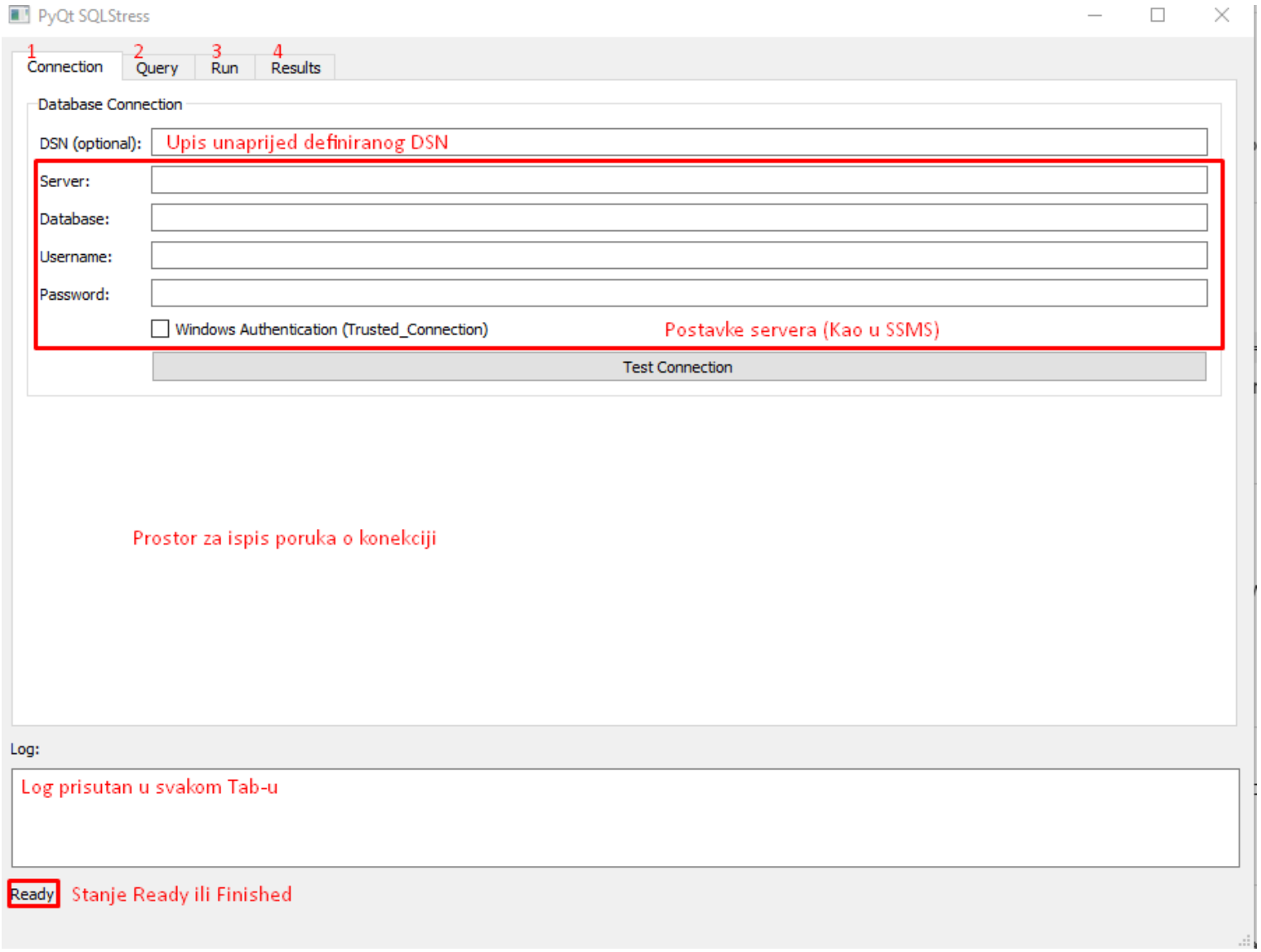
- Config- Sadrži sve parametre potrebne za jedno testiranje, npr:
  server, database, username, password, query koji će se izvršavati, broj niti (threads), broj iteracija, delay,... Ukratko: služi kao kontejner za postavke testa.
- Logger- Implementira jednostavan logger koji šalje poruke u GUI (u Log polje). Koristi Qt signale (pyqtSignal) da bi se svaka poruka koja se doda odmah prikazala u interfejsu.
- Runner- Sadrži glavnu logiku za pokretanje i izvršavanje SQL upita u više niti.
  Upravo ovdje se kreira konekcija prema bazi, šalju se upiti i mjere vremena izvršavanja.
  Na kraju vraća rezultate: broj uspješnih izvršavanja, broj grešaka, prosječno, minimalno i maksimalno vrijeme. Ukratko ovo je glavni engine aplikacije -izvršava test i vraća metrike.

Controller- Povezivanje UI i logike se vrši na način da klasa *MainController* učitava .ui fajl i povezuje dugmad sa funkcijama (npr. test konekcije, pokretanje testa, iscrtavanje rezultata).

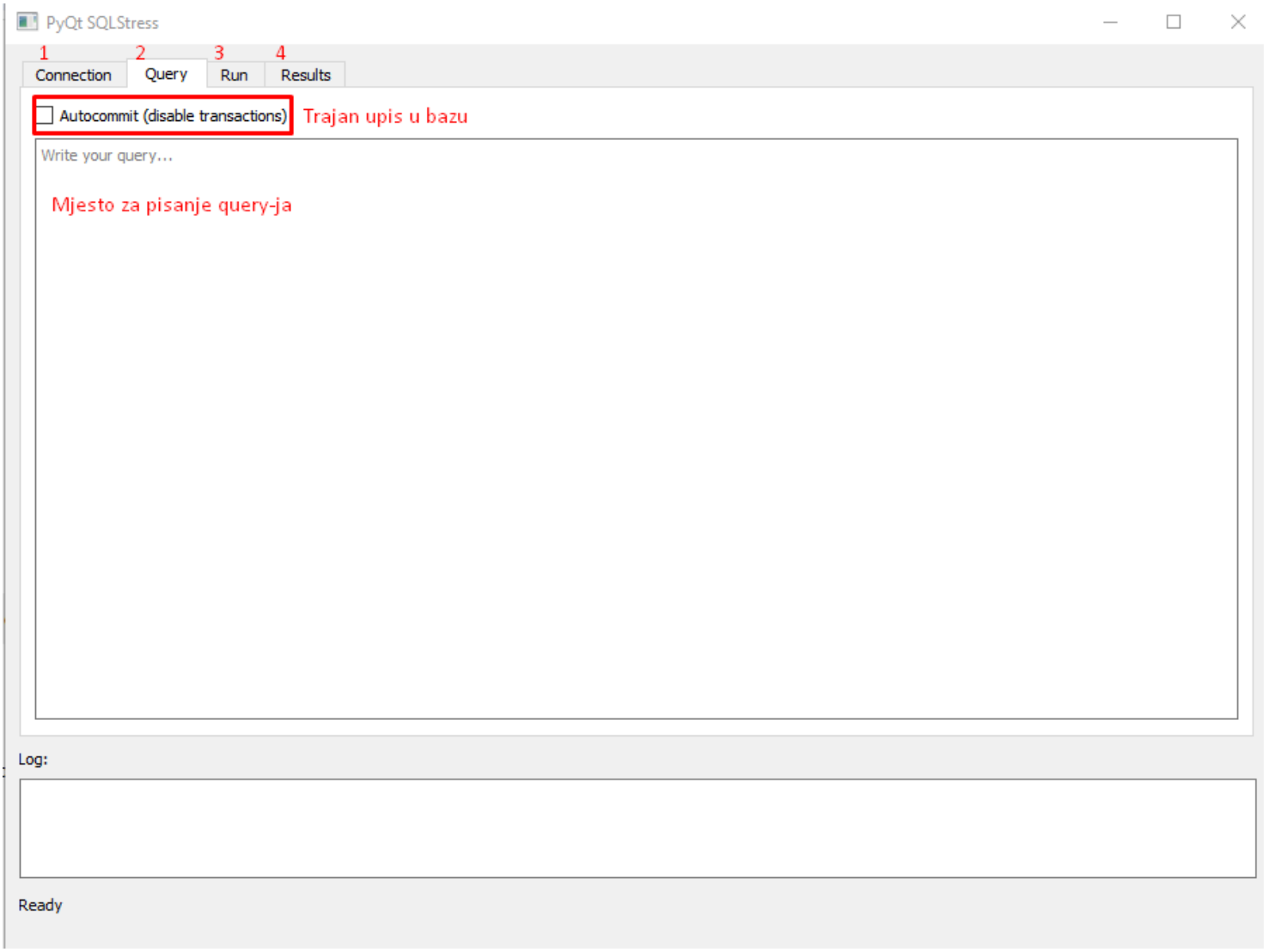
UI- Dizajn interfejsa (UI)- fajl je generisan pomoću Qt Designer-a sa četiri taba, o čijem ću izgledu i funkcionalnostima pričati u nastavku. Aplikacija posjeduje 4 prozora: Connection, Query, Run i Results. U ove prozore je rapoređena radi bolje preglednosti ali i razdvajanja koraka.

1. Connection- služi za unos podataka o bazi (server, baza, username, password) i testiranje da li se konekcija može uspješno uspostaviti.
2. Query- ovdje upisuješ SQL upit koji želiš da se izvršava tokom testiranja, te biraš opciju autocommit.
3. Run- podešavaš parametre opterećenja: broj niti (threads), broj ponavljanja (iterations) i kašnjenje između njih, te pokrećeš ili zaustavljaš test.
4. Results- prikazuje rezultate testiranja: tabelu s metrikama (prosječno, minimalno i maksimalno vrijeme izvršavanja, broj grešaka…) i grafove koji vizualizuju trajanje upita i aktivnost niti.

Svaki od ovih prozora posjeduje jedan te isti Log. Na slikama ispod su prikazani prozori, jedan po jedan, a na slikama se nalaze jednostavna objašnjenja dijelova, tj polja unutar prozora.

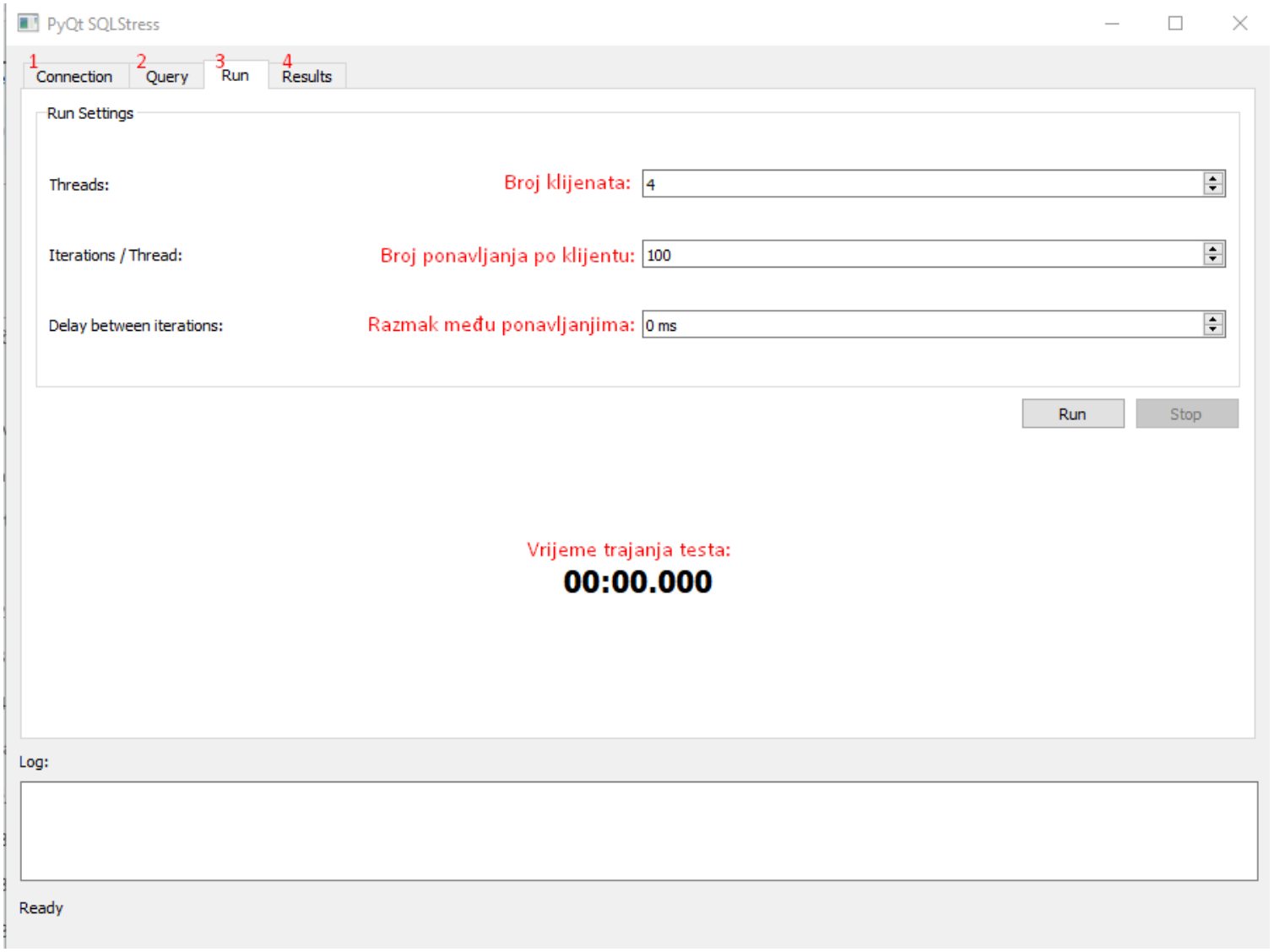


Slika 51 Connection prozor aplikacije

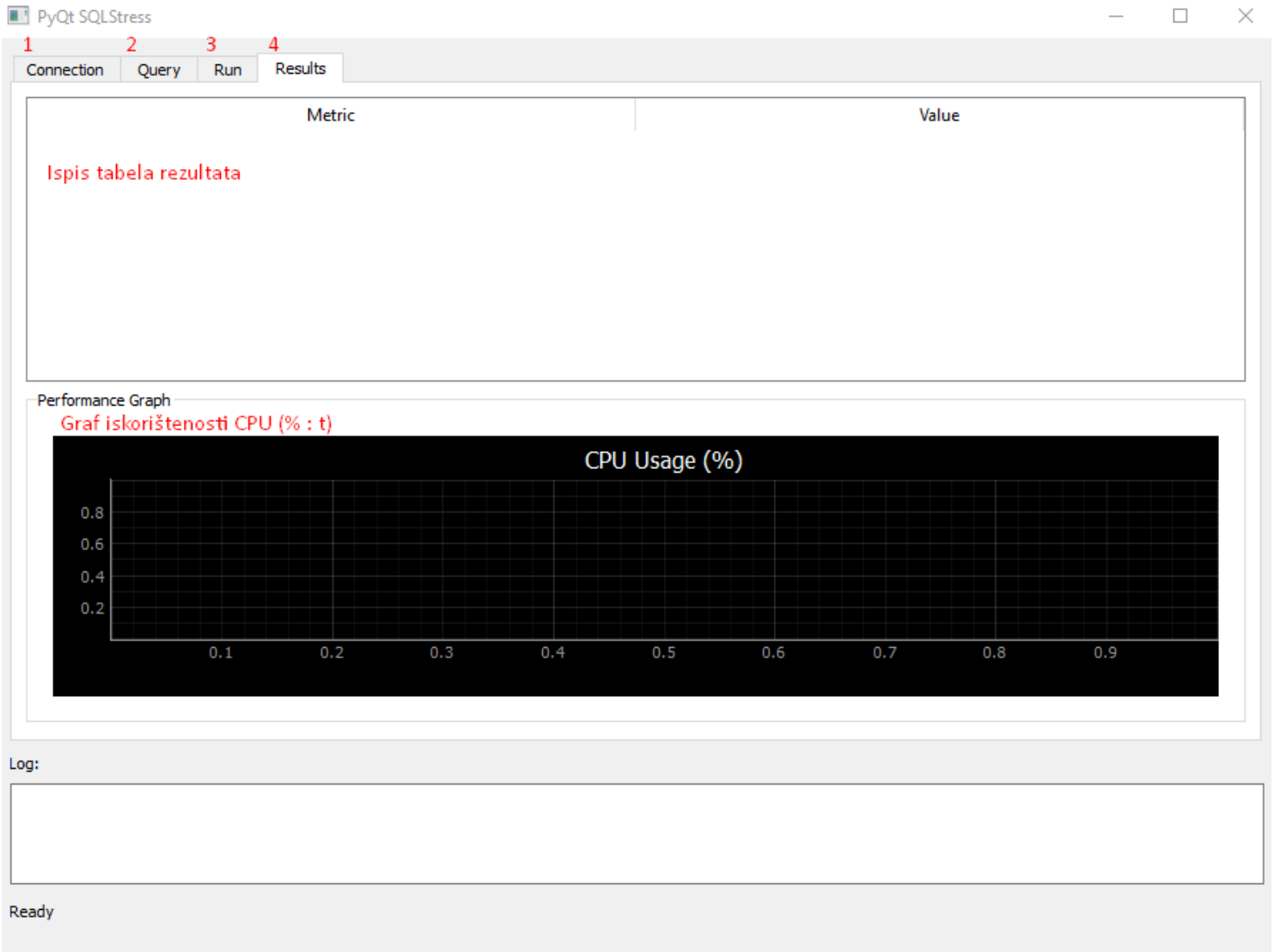


Slika 52 Query prozor aplikacije

Zanimljivo u ovom prozoru je Autocommit opcija. Moram priznati da postoji mogućnost da ova stavka nike u potpunosti ispravno implementirana. Zamisao je naredna: Ako je Autocommit uključen svaki INSERT, UPDATE, DELETE koji se testira stvarno ide u bazu i odmah se upisuje.
Dakle, ako se testira INSERT INTO users (...) sa 100 ponavljanja, stvarno ćemo imati 100 novih korisnika u tabeli.
Ako ti je Autocommit isključen, svaki upit ide u transakciju, ali se na kraju radi ROLLBACK. To znači da se promjene ne sačuvaju u bazi nakon testa. Dakle, može se testirati INSERT i vidjeti performanse, ali kad test završi u bazi neće ostati tih 100 redova.



Slika 53. Run prozor aplikacije



Slika 54. Results prozor aplikacije

7.1. Izazovi izrade PyQT SQLStress aplikacije

Velika olakšica u izradi ove aplikacije je bila originalna aplikacija SQLStress Query, koja već sadrži puno impleentiranih stvari koje sam koristila, poput konekcije na bazu, implementacije broja korisnika i zahtjeva, itd.

Ono što bih izdvojila kao najveći izazov kreiranja ove aplikacije jeste e dodavanje grafa za praćenje opterećenja procesora (CPU usage). Ova funkcionalnost nije postojala u originalnoj SQLStress aplikaciji, ali je uvedena kako bi se vizualno pratilo koliko testiranje SQL upita troši sistemske resurse. Ova stavka zahtjeva puno rada i ispod ću pobrojati glavne korake i karakteristike same funkcionalnosti.

Pristup podacima o CPU-u: Za očitavanje trenutnog opterećenja procesora korišten je modul *psutil*.

- Funkcija psutil.cpu_percent(interval=None) vraća postotak zauzeća CPU-a u trenutku poziva.
- Pošto se koristi interval=None, metoda je non-blocking i odmah vraća rezultat (ako bi se stavilo npr. interval=1, čekalo bi se cijelu sekundu).
- Na ovaj način se u pravilnim vremenskim intervalima (svakih 1000 ms) dobijaju nove vrijednosti koje se dodaju u listu (self.cpu_data).

```
def update_cpu_graph(self):
    usage = psutil.cpu_percent(interval=None)  # non-blocking
    self.cpu_data.append(usage)
    if len(self.cpu_data) > 100:  # čuvaj zadnjih 100 mjerenja
        self.cpu_data.pop(0)
    self.cpu_curve.setData(self.cpu_data)
```

Ova funkcija je "srce" CPU monitora u tvojoj aplikaciji svake sekunde (zahvaljujući QTimer) pozove se update_cpu_graph, izmjeri trenutni CPU usage, spremi ga u listu i prikaže na grafu.

Periodično ažuriranje -QTimer: Da bi se graf kontinuirano osvježavao, korišten je Qt-ov QTimer.

- Timer je podešen da svakih 1000 ms poziva metodu update_cpu_graph().
- Svaki put se u listu dodaje nova vrijednost CPU usage, a stari podaci se režu (npr. čuva se samo zadnjih 100 mjerenja) da bi graf ostao pregledan.

Crtanje grafa - pyqtgraph: Za vizualizaciju je korišten pyqtgraph, jer je brži i lakši od matplotlib-a u realnom vremenu.

- Napravljen je PlotWidget sa crnom pozadinom i bijelim gridom.
- Osi su označene: X = broj uzoraka (samples), Y = procentualno opterećenje CPU-a.
- Kreirana je linija (self.cpu_curve) kojoj se postavlja novi dataset na svakom ažuriranju.
- Rezultat je real-time line chart koji se pomjera dok aplikacija radi.

```
self.cpu_data = []
self.cpu_plot = pg.PlotWidget()
self.cpu_plot.setBackground('k')  # crna pozadina
self.cpu_plot.showGrid(x=True, y=True, alpha=0.3)  # tanke grid linije
self.cpu_plot.setTitle("CPU Usage (%)", color="w", size="12pt")

styles = {"color": "w", "font-size": "10pt"}
self.cpu_plot.setLabel("left", "Usage %", **styles)
self.cpu_plot.setLabel("bottom", "Samples", **styles)

self.cpu_curve = self.cpu_plot.plot(pen=pg.mkPen(color=(0, 255, 0), width=2))

layout = QtWidgets.QVBoxLayout(self.cpuPlotWidget)
layout.addWidget(self.cpu_plot)

self.cpu_timer = QTimer()
self.cpu_timer.timeout.connect(self.update_cpu_graph)
```

Integracija u interfejs: Graf je dodat unutar posebnog widgeta (self.cpuPlotWidget) i postavljen u layout preko QVBoxLayout. Na ovaj način je integrisan u Results tab pored ostalih rezultata testa.

Finalno, moralo se paziti da graf ne troši previše resursa sam po sebi. Pyqtgraph je odabran jer je optimiziran za brze update-e.Odlučeno je da se čuva samo posljednjih 100 mjerenja, čime se izbjeglo preveliko opterećenje memorije i renderovanja.

7.2. Lokacija i način pokretanja

Aplikacija je dostupna za pregled na linku:https://github.com/edinakaknjo/PyQT-SQLStress

Sadrži .exe fajl i izvorni kod. .exe datoteka se nalazi na lokaciji: https://github.com/edinakaknjo/PyQT-SQLStress/tree/main/PyQT_SQLStress/dist

8. ZAKLJUČAK

Provedena analiza pokazala je da različite tehnike optimizacije u SQL Serveru daju vrlo različite rezultate, zavisno od tipa upita i konteksta korištenja.

Prva hipoteza, koja se odnosila na poređenje stored procedura i ad hoc upita, nije se potvrdila. Stored procedure nisu dosljedno pružale bolje performanse.

Druga hipoteza, vezana za denormalizovane summary tabele, u potpunosti se potvrdila. Rezultati su jasno pokazali da summary tabele značajno poboljšavaju performanse.

Treća hipoteza, koja je ispitivala prednost Full-Text Searcha u odnosu na LIKE, takođe se potvrdila.

Na osnovu svih testiranja može se zaključiti da izbor metode optimizacije zavisi od konkretnog scenarija i tipa operacije, ali je evidentno da denormalizovane summary tabele i Full-Text Search predstavljaju tehnike koje daju značajna poboljšanja i mogu se preporučiti u realnim sistemima. Stored procedure, iako korisne u nekim situacijama, ne garantuju uvijek bolje performanse i njihov benefit se najviše ogleda kod ponavljanih upita gdje dolazi do izražaja keširanje execution plana

Za tetsiranje su korištene razne tehnike, a kao najbitniju želim izdvojiti vlastitu aplikaciju PyQTStress.