Napredni načini za optimizaciju performansi na SQL serveru

Kupanovac, Leon

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni diplomski studij

NAPREDNI NAČINI ZA OPTIMIZACIJU PERFORMANSI NA SQL SERVER ENGINE-U

Diplomski rad

Leon Kupanovac

Osijek, 2021./2022.

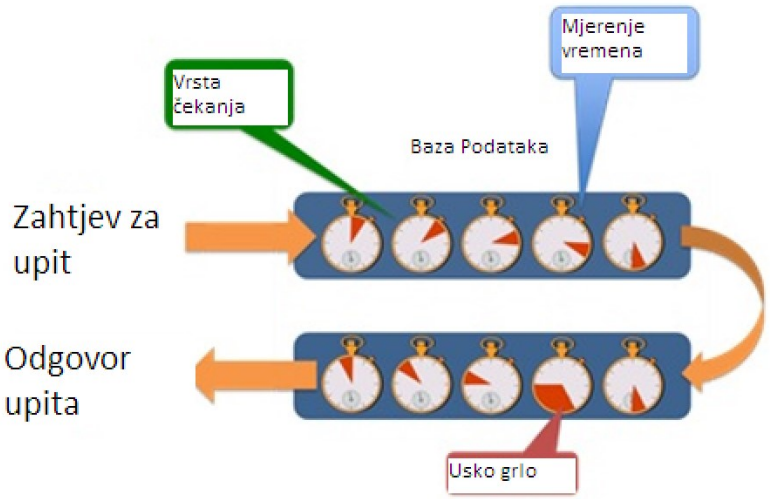
SADRŽAJ

1. UVOD

Podešavanje (eng. Tuning) baze podataka izazovan je zadatak, naročito u produkcijskim bazama jer informacije i koncepti koji podržavaju performanse trebaju sveobuhvatno i dubinsko razumijevanje programske podrške (eng. Softwarea) baze podataka i fizike sklopovlja.

Sveobuhvatno podešavanje baza podataka uključuje podešavanje same aplikacije, instanci te upita (eng. Queries). Aplikacijsko podešavanje predstavlja podešavanje dijelova programa koji čine samu aplikaciju, dok podešavanje instanci predstavlja podešavanje samog programa (eng. Engine-a) baze podataka. Podešavanje upita predstavlja dohvaćanje što više informacija u što manje vremena. Također bitno je naglasiti da se do 85% podešavanja performansi temelji na samom podešavanju upita.

Sigurnost igra glavnu ulogu prilikom upravljanja performansama, razlog tome je da se nepotrebni ili zlonamjerni zahtjevi ne izvrše. Uljezi ne mogu komunicirati sa sustavom baze podataka putem učinkovitog sustava provjere autentičnosti. Stoga sustav provjere autentičnosti mora biti robusniji što je više moguće.



Slika 1.1: Analiza vremena odaziva u modulu baze podataka[1]

Na slici 1.1 prikazani su događaje čekanja od kraja do kraja koje je doživio upit. Ukratko, događaj čekanja može biti u obliku procesa poslužitelja ili niti koja čeka da se događaj dovrši ili da resurs postane dostupan prije nego što može nastaviti s obradom upita. Premještanje podataka u među spremnik može se smatrati primjerom za događaje čekanja.

Zbog složenosti infrastrukture često je teško utvrditi temeljni uzrok problema performansi, osim ako se instrumentacija performansi ne ugradi u kod u vrijeme razvoja.

2. NAČINI ZA OPTIMIZACIJU UPITA
2.1. Indeksiranje

Indeksiranje je bez sumnje najbolji način kako bi se poboljšale performanse upita i aplikacija. U ovom dijelu rada biti će objašnjen rad indeksa, definiranje te njihovo apliciranje kao i vrste indeksa.

Iznimno je važno nakon kreiranja indeksa verificirati plan izvršavanja kako indeksi ne bi bili korišteni samo od strane upita već i na očekivani način. Na primjer može se stvoriti indeks za određeni upit, ali taj indeks može biti korišten kao operacija pod nazivom skeniranje indeksa (eng. Index scan),a ne kao pretraživanje indeksa (eng. Index seek). Skeniranje indeksa predstavlja dohvaćanje svih redova iz tablica, dok pretraživanje indeksa dohvaća odabrane redove tablice. Također skeniranje indeksa dohvaća svaki red nebitno na njegovu kvalifikaciju da zadovoljava određene uvijete, i njegov trošak korištenja proporcionalan je broju redova tablice. On je najefektivnija metoda ukoliko se upiti određuju na manjim tablicama. Pretraživanje indeksa dohvaća redove koji zadovoljavaju postavljene uvijete, njegov trošak je proporcionalan broju uvjetovanih redova i stranica umjesto broju ukupnih redova tablice. Pretraživanje indeksa dijeli se na dvije vrste indeksa, a to su klasterirani indeksi (eng. Clustered index) i ne klasterirani indeksi (eng. Non-Clustered index) [2].

Očito je da su mjerni podaci o performansama glavni pokazatelj uspjeha strategije indeksiranja. Ako je izvođenje upita prvotno trajalo jednu minutu i ako se nakon stvaranja potrebnih indeksa pokrene u roku jedne do dvije sekunde može se zaključiti da je izabrana uspješna strategija indeksiranja.

Važno je napomenuti da samo zato što optimizator upita odabire indeks, to ne znači da će se dobiti dobre performanse. Navigacija indeksom b-stabla je skupa operacija i to se samo isplati pri traženju nekoliko redaka ili relativno malog broja redaka. Uobičajeno je da u optimizaciji upita postoji prekretnica u kojoj se trošak povećava korištenjem drugih metoda pristupa ili operacija, kao što je skeniranje tablice [2]. Činjenica da indeks ne pokriva sve stupce koje zahtijeva upit je razlog povećanja troškova i smanjivanja performansi jer se potencijalno treba kretati strukturom b-stabla dva puta, jedan za indeks bez kontrole i drugi za glavnu tablicu, obično grupirani indeks. Takva se operacija naziva pretraživanje ključa (eng. Key lookup). Glavna tablica također može biti hrpa gdje nema b-stabla, ali slična operacija pretraživanja reda identiteta (eng. RID-Row ID lookup) će biti potrebna.

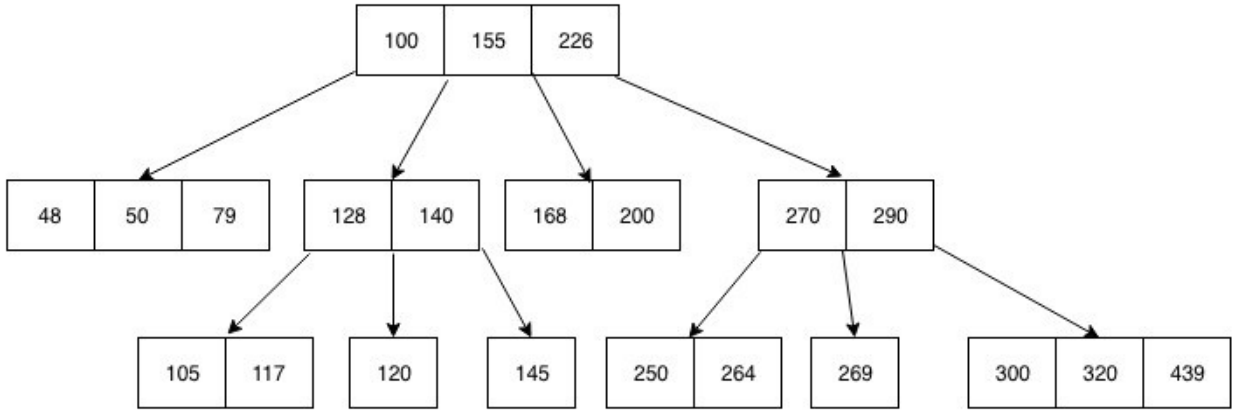
Optimizator upita dizajniran je da prati prijašnje navedene procese, ali ponekad može doći do problema, možda zbog nekih loših procjena kardinalnosti ili nekih drugih ograničenja kod procesorskog upita. Osim toga neki ponovno iskoristivi kôd kao spremljene procedure mogu stvoriti problem gdje su neki upiti osjetljivi na parametre posebno u uvjetima kod neravnomjerne distribucije podataka. U tom slučaju optimizator upita kreira optimalan plan gdje se spremljene procedure izvršavaju prvi puta. To se postiže korištenjem prvog dostupnog parametra ili parametra koji donose odluku o korištenju operacija prilikom izvršavanja takvog upita [2]. Problem se može dogoditi poslije, kod takvih upita gdje se pokreće druga instance s različitim parametrima u kojima prethodni plan više nije dobar odabir za poboljšanje performansi.

Konačno, samo postojanje WHERE uvjeta u upitu znači da indeks mora biti kreiran ili mora biti korišten od strane optimizatora upita, ako on već postoji. Na kraju krajeva ovo je odluka temeljena na troškovima i u nekim slučajevima ostala navigacija po odlukama može biti jeftinija i prihvatljivija. Tipični primjer ovoga su upiti koji nisu dovoljno selektivni ili ne pokrivaju cijeli upit. Postoji nekoliko razloga zašto indeks nije odabran od strane optimizatora upita, čak i pod pretpostavkom da je indeks ispravno stvoren.

- **Trošak.** Možda je korištenje indeksa dobra ideja za trenutnu vrijednost ili parametar, ali ne i za neke druge vrijednosti. Taj se problem događa s podacima s neravnomjernom raspodjelom.
- **Složeni izrazi.** Može se dogoditi da zbog korištenja složenih funkcija ili složenih izraza optimizator ne može u potpunosti razumjeti upit.
- **Nije korisno.** Možda indeks uopće nije koristan. To se može istražiti zašto i izbrisati ga. Korištenje indeksa bilježi se na sys.dm_db_index_usage_ statistike DMV (eng. Dynamic Management Views).
- **Loše procjene kardinalnosti** ili neka druga ograničenja procesora upita. Ako se nakon analize ovih izbora otkrije da bi indeks zapravo mogao biti koristan i iz nekog razloga ga optimizator upita ne odabire, može se istražiti zašto ponašanje upita koristi savjet (eng. Hint) za prisiljavanje takvog indeksa. Obično se ne preporučuje korištenje savjeta jer razbijaju deklarativnu specifikaciju upita, ograničavaju izbore koje ima optimizator upita i otežavaju održavanje upita.

2.1.1. B-stablo

B-stablo predstavlja strukturu podataka koji pružaju sortirane podatke, dopuštaju pretrage, sekvencijalni pristup, priloge, te brisanje sortiranim poretkom. B-stablo je vrlo korisno kada se radi sa sustavima pohrane koji zapisuju velike blokove podataka. B-stablo pojednostavljuje binarno pretraživanje tako da dopušta čvorove s više od dvoje djece [3]. Primjer B-stabla vidljiv je na slici 2.1.



Slika 2.1: B-stablo[4]

B-stablo sprema podatke tako da svaki čvor sadrži ključ ograničenja (eng. Constraint Key), uzlaznim redoslijedom. Svaki ključ ima dvije reference za druga dva dijete čvora. Na slici 2.1. vidljivo je da su ključevi podređenog čvora lijeve strane manji od trenutnih ključeva, također vidi se da su s desne strane veći. Ako jedan čvor ima „*n*“ broj ključeva tada može imati najviše „*n+1*“ podređenih čvorova.

Struktura B-stabla sastoji se od tri dijela. Prvi je razina korijena (eng. Root level). On sadrži pokazivače koji pokazuju na podatke razine listova (eng. Leaf level). Ako broj lišća premašuje broj pokazivača tada se implementira sloj grana (eng. Branch level). Broj slojeva grana ovisi o količini podataka spremljenih u indeks [4].

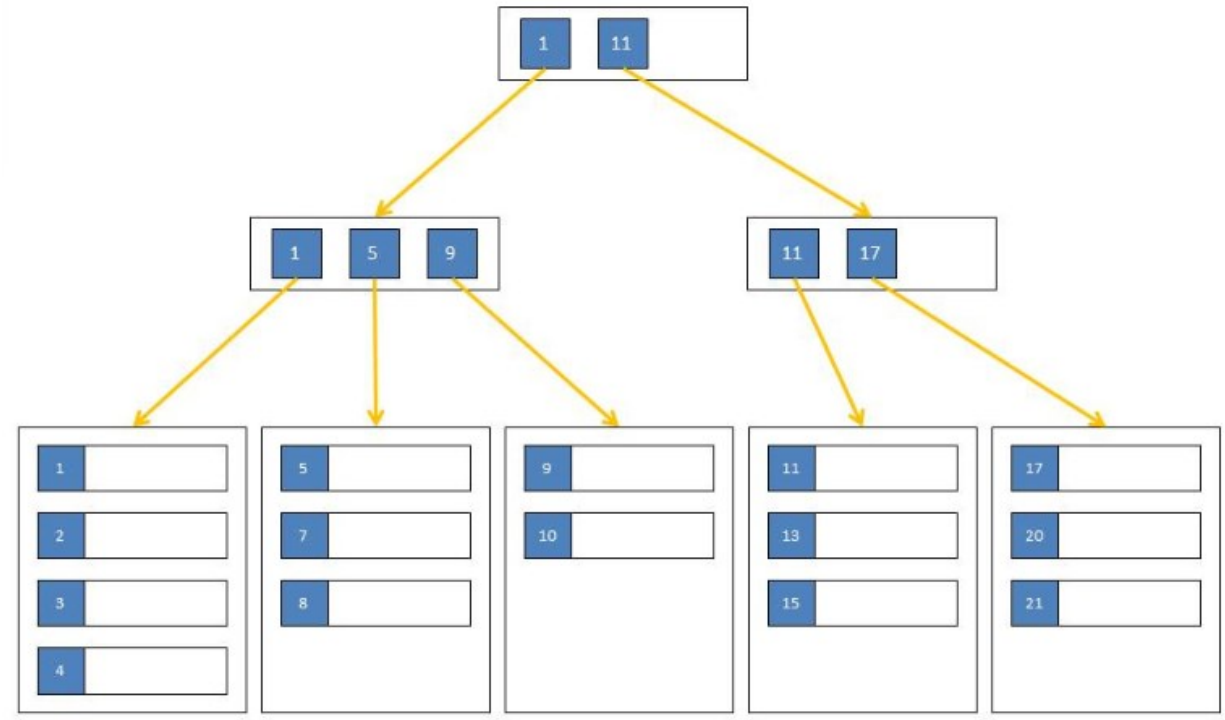
SQL Server indeksi sadrže veliki broj čvorova na svakoj razini. To pomaže pri povećanju efikasnosti prilikom kreiranja indeksa tako da se izbjegne potreba za velikom dubinom. Dubina indeksa predstavlja broj razina od sloja korijena do sloja lišća [5]. Indeks koji je dubok imat će problema s performansama kao i s degradacijom. U suprotnom indeks s velikim brojem čvorova na svakoj razini imat će vrlo ravnu strukturu indeksa. Indeks sa tri do četiri razine je uobičajen.

Isto tako postoje još dodatna mjerila za indekse. Jedno od njih je i gustoća indeksa. Ona predstavlja mjeru unikatnosti podataka u tablici. Gust stupac je onaj stupac koji ima puno istih redova odnosno duplikata. Drugo svojstvo je selektivnost indeksa. Selektivnost indeksa mjeri

koliko redova je skenirano u odnosu na ukupan broj redova. Indeks koji ima veliku selektivnost znači da je mali broj redova skeniran u odnosu na ukupan broj redova.

2.1.2. B+stablo

B+stablo ima puno veću efikasnost od binarnog stabla kada je u pitanju ažuriranje podataka, ali neke operacije mogu i dalje biti dosta skupe ovisno o čvoru koji sadrže nove ili ažurirane podatke u stablu. Umjesto da se sve čvorove tretira jednako, nova struktura ima dvije vrste čvorova. Najniži sloj koji sadrži podatke (sloj lišća). Svi ostali čvorovi uključujući i čvorove korijena sadrže samo ključne vrijednosti i pokazivače na sljedeće čvorove. Ova vrsta stabla naziva se B+Stablo [6]. Primjer B+Stabla vidljiv je na slici 2.3.



Slika 2.3: B+stablo [7]

Ne postoje ograničenja na broj parova ključeva i pokazivača ili broj redova podataka unutar jednog čvora. Jedino ograničenje je da svi čvorovi lišća imaju istu udaljenost od čvora korijena. To znači da je vrijeme i rad za pretraživanje određene točke uvijek isti bez obzira na vrijednost ključa. Također B+Stablo uvijek čuva ažuriranja koja su lokalizirana u stablu. Možda će morati premjestiti nekoliko redaka podataka u novi čvor tijekom ažuriranja, ali će vjerojatno trebati promijeniti samo jedan nadređeni čvor kako bi se integrirao taj novi čvor. Postoji mogućnost da jedna promjena utječe na svaku razinu stabla, ali u praksi takve primjene su rijetke [6].

SQL Server svoje indekse sprema u B+Stablo formatu. Postoji nekoliko iznimki npr. kada se radi o privremenim (eng. Hash) indeksima koji se kreiraju prilikom HASH JOIN operacije ili kod COLUMN STORE indeksa koji uopće i nisu indeksi. Svi ne klasterirani i klasterirani indeksi bazirani su na B+Stablu.

Svaki čvor unutar B+Stabla predstavlja stranicu u SQL Serveru. Postoje dvije vrste stranica. Prvi tip stranica su podatkovne. U podatkovnim stranicama svaki čvor razine lista u SQL Server B+Stablu je jedna podatkovna stranica. Drugi tip stranica su među indeksirane stranice. U tom tipu svaki čvor u indeksu B+Stabla koji nije čvor razine lista predstavlja jednu stranicu tog tipa. Te stranice sadrže redove baš kao i podatkovne stranice, ali osim toga sadrže i pokazivač za svaki redak koji identificira sljedeću podređenu stranicu. Ta dijete stranica može biti tip 1 ili tip 2 ovisno o lokaciji u B+Stablu.

2.1.3. Održavanje indeksa

Indekse nije potrebno kreirat za svaki upit u sustavu. Oni se preporučaju kao potpora performansama upita koji se često izvršavaju, ako imamo previše indeksa unosimo nove probleme s performansama u sustav. Svaki indeks zahtjeva održavanje bilo to automatsko održavanje (svakim korištenjem naredbi: INSERT, UPDATE, DELETE) ili od strane administratora koji planira poslove (eng. Jobs) kako bi se uklonila fragmentacija ili ažurirala statistika. Indeksi također koriste najbitniji resurs, a to je prostor za pohranu te memorijske resurse. Broj indeksa koji se kreira na tablici ovisi imamo li OLTP (eng. Online Transactional Processing) sustav gdje se podaci mijenjaju cijelo vrijeme ili skladište podataka (eng. Data Warehouse) gdje se podaci mijenjaju kroz neki period vremena npr. korištenje ETL (eng. Extract Transform Load) paketa. Karakteristike indeksa isto tako ovise i o kompleksnosti upita odnosno koristili se indeks za pronalazak jednog podataka ili kako bi potpomogao agregaciji nad podacima [8]. Glavna 3 problema korištenja previše indeksa su sljedeća [9]:

a) Slabije performanse operacija ažuriranja poput INSERT, UPDATE, DELETE, itd.

b) Usporavanje vremena poslova održavanja prilikom ponovne izgradnje indeksa ili reorganizacijskih operacija (operacije koje potpomažu fragmentaciji indeksa), te ažuriranje statistike.

c) Veća iskoristivost prostora za pohranu i memorijskih resursa.

2.1.4. Klasterirani indeksi

Klasterirani indeks je indeks koji definira fizički poredak u kojem su zapisi tablice pohranjeni u bazi podataka. Budući da može postojati samo jedan način na koji se zapisi fizički pohranjuju u tablicu baze podataka, može postojati samo jedan klasterirani indeks po tablici. Prema zadanim postavkama na stupcu primarnog ključa stvara se klasterirani indeks [8].

Zato što hrpa ne predaje nikakav red i zahtjeva neklasterirani indeks kako bi se mogao pronaći specifičan red, klasterirani indeks ima prednost da ide u korist tome, zato što može postojati samo jedan klasterirani indeks on se mora pažljivo odabrati.

Najbolji odabir klasteriranog indeksa je onaj koji može pomoći sa velikom većinom upita gdje indeks može iznimno brzo pronaći prvi red u nekom rasponu. Upiti u nekom rasponu su upiti koji filtriraju rezultate koji koriste operatore poput >, >=, <, ili <= kao i BETWEEN operator [8].

Klasterirani indeksi također pomažu pri JOIN operaciji, npr. kada definiramo glavnu tablicu sa stranim ključevima. Najviše koristi od njih imaju ORDER BY i GROUP BY upiti ako su bazirani na klasterirani ključ.

Najbolja praksa pokazala je da klasterirani indeks treba biti unikatan, statičan, uzak posebno kod razmatranja performansi i prostora na disku. Kao dodatna napomena navedene četiri preporuke nisu problem hrpi jer redni ID (RID) (eng. Row ID) slijedi ta četiri svojstva. RID je relativno uzak jer koristi samo osam bajtova. Četiri svojstva koja se preporučaju za klasterirani indeks su:

Jedinstvenost. Preporuča se da je ključ klasteriranog indeksa jedinstven, ako nije 4 bajta jedinstvenifikatora će se dodati kako bi on bio jedinstven.

Uskost. Preporuča se da klasterirani ključ bude što uži zbog uštede prostora na disku te korištenje memorije. Svaki neklasterirani red indeksa također će spremati klasterirani ključ koji također utječe na prostor na disku i korištenje memorije. Klasterirani ključ može biti jedan ili više stupaca, stoga se preporuča da bude što je manje stupaca moguće ili jedan stupac i jedan kratki ključ.

Statičnost. Ovo svojstvo bi trebalo biti zahtjev za klasterirani ključ. Nepostojani klasterirani ključ (eng. Volatile clustering key) koji se može konstantno mijenjati može predstavljati veliki problem na performanse zato što se rezultati moraju micati sa jedne stranice na drugu i kreirati raspodjelu stranica i fragmentaciju.

Preporuča se korištenje sve veće vrijednosti poput one koje pruža svojstvo identiteta. To pomaže u dodavanju novih redaka u nove stranice na kraju indeksa kako bi se izbjegle podjele stranica i fragmentacija. Međutim u vrlo intenzivnim radnim opterećenjima s visokom konkurentnošću mogu se stvoriti scenarij gdje se sav sadržaj umeće na istoj i posljednjoj stranici. Ovaj problem je poznat kao sukob umetanja zadnje stranice (eng. Last page insert contention) koji također može utjecati na hrpe [3].

2.1.5. Ne klasterirani indeksi

Neklasterirani indeks predstavlja tradicionalni sekundarni indeks koji koristi SQL Server i može biti kreiran na glavni klasterirani indeks ili na hrpi. Neklasterirani indeksi sadrže samo jedan ili više stupaca bazne tablice koji se ponašaju kao indeks ključevi. Neobavezno mogu sadržavati jedan ili više stupac ne kao ključ nego kao dodatne podatke kako bi pokrili široki spektar upita koristeći INCLUDE naredbu [8].

Neklasterirani indeksi neophodni su za upite u kojima se traži potpuno poklapanje jednog ili više redova korištenjem usporedbe jednakosti. Kao klasterirani indeksi, neklasterirani indeksi mogu biti ekstremno korisni za upite ograničenog spektra, ali samo ako indeks pokriva rezultate cijelog upita.

2.1.6. Filtrirani indeksi

Filtrirani indeksi uvedeni su sustavom SQL Server 2008 i mogu biti od velike pomoći u situacijama u kojima se pod skupovi podataka mogu eliminirati iz rezultata upita. Tipični scenariji za filtrirane indekse su podaci s velikim brojem NULL vrijednosti ili stupce s heterogenim podacima. Filtrirani indeksi zahtijevaju uvjet WHERE s predikatom filtra koji označava retke koje treba uključiti u indeks [10].

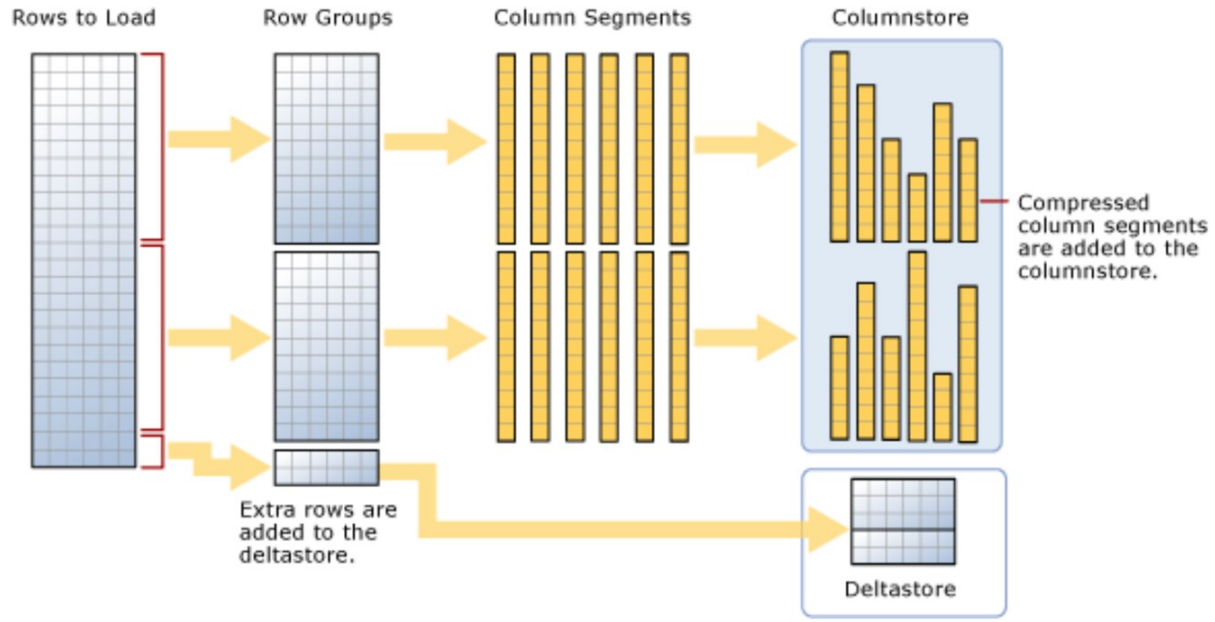
2.1.7. Columnstore indeksi

Columnstore indeksi prvi su put uvedeni u SQL Server 2012. Oni su novi način pohrane podataka iz tablice koji poboljšava performanse određenih vrsta upita za najmanje deset puta. Posebno su korisni s tablicama činjenica (eng. Fact tables) u skladištima podataka. Kada su Columnstore indeksi prvi put uvedeni, nije bilo moguće ažurirati tablicu Columnstore indeksom bez da su se prethodno uklonili. S vremenom je došlo mnogo poboljšanja. Ovi indeksi su vrlo korisni za radna opterećenja skladišta podataka i velike tablice [11]. U nekim slučajevima mogu poboljšati performanse upita (eng. Query-a) za faktor 10, pa je znanje i razumijevanje načina na koji funkcioniraju ključno ako se radi o okruženju s većim podacima.

Columnstore jednostavno znači novi način pohrane podataka u indeksu. Umjesto normalnih indeksa B-Stabla ili Rowstore u kojima su podaci logički i fizički organizirani i pohranjeni kao tablica s recima i stupcima, podaci u Columnstore indeksima fizički se pohranjuju u stupce i logički organiziraju u retke i stupce.

Umjesto spremanja cijelog retka ili redaka na stranicu, na toj se stranici sprema jedan stupac iz više redaka. Upravo ta razlika u arhitekturi daje Columnstore indeksu vrlo visoku razinu kompresije, zajedno sa smanjenjem otiska skladištenja i pružanjem masovnih poboljšanja u performansama čitanja [12].

Indeks funkcionira rezanjem podataka na segmente koji se mogu komprimirati. Potrebna je grupa redaka, najmanje 102.400 redaka s najviše oko milijun redaka, nazvana grupa redaka, a zatim mijenja tu grupu redaka u segmente stupaca. Upravo su ti segmenti osnovna jedinica pohrane za Columnstore indeks.



Slika 2.4: Proces komprimiranja pomoću Columnstore indeksa [12]

Primjer je tablica s 2,1 milijun redaka i šest stupaca. Što znači da daljnjom obradom postoje dvije grupe redaka od po 1.048.576 redaka i ostatak od 2848 redova, koji se naziva delta grupa. Budući da svaka grupa redaka sadrži najmanje 102.400 redaka, delta grupa redaka koristi se za spremanje svih preostalih zapisa indeksa dok ne dobije dovoljno redaka za stvaranje druge grupe redaka [12]. Može imati više delta grupa redaka koje čekaju da budu premještene u kolone. Više delta grupa pohranjeno je u delta skladište (eng. Delta store), a to je zapravo indeks

B-stabla koji se koristi uz Columnstore. U idealnom slučaju, indeks će imati grupe redaka koje sadrže blizu milijun redaka što je više moguće kako bi se smanjili takozvani režijski troškovi operacija skeniranja.

Postoji proces koji se pokreće da bi premjestio delta grupa redaka iz delta skladišta u Columnstore indeks koji se naziva *tuple - mover* proces. Ovaj proces provjerava zatvorene grupe, što znači grupu koja ima najviše milijun zapisa i spremna je za komprimiranje i dodavanje u indeks. Kao što je prikazano na slici 2.4, indeks stupca sada ima dvije grupe redaka koje će zatim podijeliti u segmente stupaca za svaki stupac u tablici. Time se stvara šest stupova od milijun redaka po grupi redaka za ukupno 12 segmenata stupaca. Upravo se ti segmenti stupaca komprimiraju pojedinačno za pohranu na disku. Program uzima ove stupove i koristi ih za vrlo visoko paralelno skeniranje podataka. *Tuple - mover* postupak se može prisiliti i ponovnim pokretanjem Columnstore indeksa [12].

Da bi olakšali brži pristup podacima, u zaglavlju stranice pohranjuju se samo minimalne i maksimalne vrijednosti za grupu redaka. Osim toga, obrada upita, kao što se odnosi na spremište stupaca, koristi Batchmode omogućujući programu da obrađuje više redaka odjednom. To također čini program sposobnim obraditi redove izuzetno brzo u nekim slučajevima, dajući dva do četiri puta veću izvedbu jednog postupka upita [12]. Na primjer, ako se radi pridruživanje odnosno agregacija, to se događa vrlo brzo jer se samo redak koji se agregira čita u memoriju i pomoću grupa redaka program može skupno obraditi grupe od milijun redaka. U sustavu SQL Server 2019 skupni način rada također će se uvesti u neke indekse spremišta redaka i planove izvršavanja.

Još jedna bitna, ali i zanimljiva razlika između Columnstore indeksa i indeksa b-stabla je u tome što indeksi stupaca nemaju ključeve. Također mogu se dodati svi stupci koji se nalaze u tablici, sve dok nisu ograničena vrstom podataka, u Columnstore indeks koji nije grupiran i ne postoji koncept uključenih stupaca. Ovo je radikalno nov način razmišljanja ako se koristi podešavanje tradicionalnih indeksa.

2.2. Poboljšanje performansi koristeći statistiku čekanja

Kada korisnička aplikacija pošalje SQL Serveru zahtjev za podacima, glavni element ukupnog odziva sustava u idealnom slučaju bi bilo vrijeme obrade CPU-a. Drugim riječima to je vrijeme koje treba CPU da dohvati, pripremi, sortira podatke i prikazuje ih ovisno u početnom upitu. U prometnom sustavu baze podataka sa stotinama i tisućama zahtjeva korisnika koji se nadmeću za ograničene resurse servera baze podataka, postojat će vrijeme kada će neki zahtjev biti stavljen na čekanje [9].

Svaki put kada je zahtjev stavljen na čekanje, SQL Server sprema dužinu trajanja čekanja, razlog postojanja čekanja, te vrstu čekanja koja generalno govori na koje resurse zahtjev čeka. To se naziva statistikom čekanja (eng. Wait statistics).

2.2.1. Raspored upravljanja resursima

Raspored upravljanja resursima može biti opisan kao dio programske podrške koji koordinira izvršavanjem različitih procesa, te upravlja sa dostupnim resursima. SQL Server ima svoj vlastiti mehanizam upravljanja resursima, on se također naziva SQLOS. Razlog njegovog postojanja je da nadomjesti Windows schedulera, odnosno Windows ne može zadovoljiti relacijske baze podataka [9]. Windows koristi mehanizam preventivnog planiranja. SQL Server koristi kooperativni mehanizam planiranja gdje procesorske niti mogu dobrovoljno doprinijeti optimizaciji. To omogućuje SQL Serveru da optimizira iskoristivost procesora, jer kada je nit signalizirana za izvršavanje, ali nije spremna za pokretanje, ona može svoj dio vremena dodijeliti u korist drugih procesorskih niti.

SQL Server sadrži po jednog schedulera za svaku procesorsku jezgru neovisno radi li se o fizičkoj ili hyperthread-iranoj jezgri, ali scheduleri nisu vezani na procesor osim ako je to definirano afinitetnom maskom. To znaci da scheduler koji radi na prvom procesoru promjenom može završiti na drugom procesoru [13].

Nit može bit u jednom od tri stanja kako je pokazano u stupcu sys.dm_exec_request DMV:

- RUNNING – na procesoru.
- SUSPENDED – kada nit zahtjeva resurs kao npr. stranicu koja nije u memoriji, tada nit predaje procesoru i prelazi na listu čekanja sa statusom niti SUSPENDED dok resurs ne bude dostupan.

- RUNNABLE – ako nit nije na čekanju resursa, i nije predana procesoru, budući da scheduler kojem je dodijeljena trenutno ima pokrenutu drugu nit, tada će ta nit biti smještena u FIFO (eng. First In First Out) red čekanja.

2.2.2. Korištenje analize čekanja za tuniranje performansi

Analiza čekanja statistike predstavlja vrlo efektivnu metoda za dijagnosticiranje vremena odaziva u bazi podataka. Vrlo jednostavno rečeno zahtjevi ili rade ili su u stanju čekanja [13].

Vrijeme odziva = vrijeme izvršavanja + vrijeme čekanja

Normalna je pojava da niti čekaju za vrijeme izvršavanja. Zapravo u jako velikom i zauzetom sustavu baze podataka gotovo je neizbježno da nit mora čekati na resurse u nekom dijelu izvršavanja. Statistika čekanja nije pravilan alat ukoliko se on koristi samostalno kako bi se dijagnosticirao problem pri performansama. Nju je najbolje koristiti u suradnji s drugim alatima za dijagnosticiranje, kako bi što manje vremena potrošili na nepotrebno rješavanje problema.

2.2.3. Istraživanje aktivnih, ali blokiranih zahtjeva pomoću sys.dm_os_waiting_tasks

Profil čekanja u stvarnom vremenu SQL Server instance moguće je ispitati u smislu zahtjeva i sesija povezanih s nitima koje su suspendirane tako što možemo postaviti upit na sys.dm_os_waiting_tasks DMV i povezati taj upit sa drugim korisnim pogledima i funkcijama[9]. sys.dm_os_waiting_tasks DMV nam govori sljedeće:

- **session_id** sesije koja je povezana sa suspendiranom niti.
- **execution_context_id** od zadatka koji je asociran sa suspenzijom ako postoji više od jednog execution_context_id-a asociranog s jednim session_id-jem.
- **wait_type** – trenutna vrsta čekanja suspendirane niti.
- **wait_duration_ms** – duljina čekanja u jedinici vremena odnosno u milisekundama, za koje vrijeme je nit koja je suspendirana čekala.
- **blocking_session_id** – prikazuje session_id od blokirajuće niti, ako je nit suspendirana kao rezultat blokiranja.

- **resource_description** – za neke vrste resursa sami opis resursa suspendirane niti je potreban npr. ako se dogodi LOCK WAIT, ovaj opis nam govori na kojem ID-u, stranici, tablici se taj lock dogodio.

U takvim slučajevima biti će vjerojatno primijećeno značajno čekanje na lock (gdje je vrsta lock-a LCK_M_XX), što ukazuje na to da sesija ne može nastaviti sa svojim radom dok ne dohvati lock na resursu na kojeg druga sesija niti drži lock. Najčešći uzrok konstantnog blokiranja je loše napisan kôd, koji omogućuje SQL Serveru da drži lock duže od potrebnog vremena, također manjak indeksa i intenzivni IO procesi mogu blokirati druge sesija na duže periode [9]. Primjer korištenja upita za statistiku čekanja vidljiv je na slici 2.5.

```
SELECT
          blocking_session_id =  blocking.session_id
        , blocked_session_id = blocked.session_id
        , wait_type = waitstats.wait_type
FROM sys.dm_exec_connections blocking
INNER JOIN sys.dm_exec_requests blocked  ON blocked.blocking_session_id = blocking.session_id
CROSS APPLY sys.dm_exec_sql_text(blocked.statement_sql_handle) blocked_cache
CROSS APPLY sys.dm_exec_sql_text(blocking.most_recent_sql_handle)
INNER JOIN sys.dm_os_waiting_tasks waitstats ON waitstats.session_id = blocked.session_id
```

Slika 2.5: Primjer upita za korištenje statistike čekanja

2.2.4. Analiziranje povijesne statistike čekanja pomoću sys.dm_os_wait_stats

Ako se instance SQL Servera izvodi neko duže vrijeme onda je podvrgnuta značajnoj promjeni kao npr. dodavanje novog važnog indeksa, čišćenje starog cache-a statistike čekanja, kako bi se spriječio utjecaj stare statistike na nove promjene [9]. Sys.dm_os_wait_stats DMV daje nam listu svih različitih vrsta čekanja na koje je nit naišla, kao i broj koliko su puta čekali da resurs bude dostupan i koliko se vremena čekalo. Dostupni su nam sljedeći stupci:

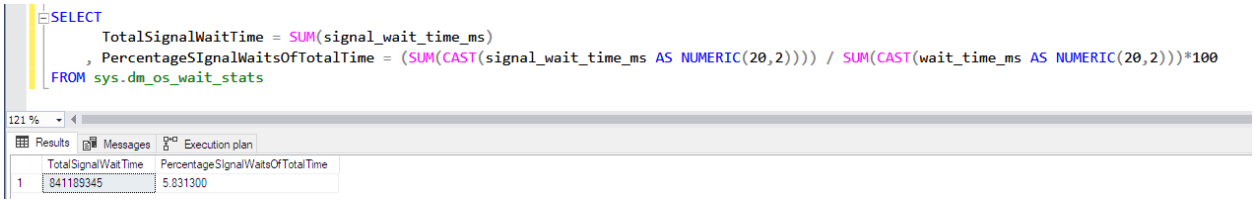
- wait_type – vrsta čekanja, uobičajeno govori o resursu na kojeg je nit stavljena u stanje čekanja (radi se o lock, latch disk IO waits itd.).
- wait_time_ms – ukupno kumulativno vrijeme koje nit mora čekati na povezanoj vrsti čekanja. Ova vrijednost uključuje signal_wait_time_ms, i ta vrijednost se povećava od trenutka kada se zadatak prestane izvršavati kako bi čekao na resurse do točke kada se nastavlja njegovo izvršavanje.
- signal_wait_time_ms – ukupno kumulativno vrijeme koje je potrebno nitima za početak izvršavanja nakon signalizacije.

- waiting_tasks_count – kumulativni ukupni broj čekanja koja su se dogodila za neki resurs (wait_type).
- max_wait_time_ms – maksimalno vrijeme kašnjenja niti.

Mnogo je razloga zašto će određeni zadatak unutar SQL Servera možda morati pričekati, što znači da postoji mnogo mogućih vrijednosti za stupac wait_type.

2.2.5. Identificiranje visokih signala čekanja

Ako je vrijeme čekanja signala značajni dio ukupnog vremena čekanja to znaci da zadaci čekaju relativno dugo vremena kako bi se nastavili izvršavati nakon što su resursi koje su čekali postali dostupni. To može ukazivati na to da postoji mnogo upita s intenzivnim korištenjem procesora kojima je potrebna optimizacija ili da server treba više snage procesora. Upit na slici 2.6 mjeri koliko je vrijeme čekanja signala u odnosu na ukupno vrijeme čekanja.



Slika 2.6: Upit mjerenja vremena čekanja signala u odnosu na ukupno vrijeme čekanja

2.2.6. Tipovi čekanja

U globali dok se promatra statistika čekanja nebitno bila ona povijesna ili od trenutno aktivnih zahtjeva bitno je fokusirati se na zahtjeve sa najviše čekanja koji su povezana sa sljedećim specifičnim vrstama čekanja [9].

- **CXPACKET**

Često ukazuje na ništa više nego da se određeni upit izvršava paralelizmom. CXPACKET čekanja u server nisu neposredan znak problema, ali mogu biti simptom drugog problema povezanog s jednom od drugih vrsta čekanja visoke vrijednosti na instanci [14].

- **SOS_SCHEDULER_YIELD**

Kada se dogodi ova vrsta čekanja konstantno, trebalo bi istražiti i pronaći dokaze koji ukazuju na to da je server pod velikim zauzećem procesora [9].

- **THREADPOOL**

Kod ovog čekanja zadatak je morao pričekati da bi se na njega povezala radnička nit kako bi se on izvršio. To bi mogao biti znak "*gladovanja*" radničke niti (eng. Worker thread starvation), koja zahtjeva povećanje broja procesora na server. Također može biti znak blokiranja što rezultira velikim brojem paralelnih izvođenja zadataka koji duže vrijeme troše radničke niti [8].

- **LCK_***

Ova vrsta čekanja događa se kada se zahtjev čeka na zajedničko zaključavanje. To se obično događa kada su zahtjevi za čitanje blokirani zahtjevima koji su otvoreni dulje vrijeme [14].

- **PAGEIOLATCH_*, IO_COMPLETION, WRITELOG**

Navedena čekanja su najčešće asocirana sa I/O disk bottleneck-ovima, iako je glavni uzrok ovog problema loš upit koji troši velike količine memorije na poslužitelju ili jednostavno nema dovoljno memorija za među spremnik [8].

- **PAGELATCH_***

Page latchevi koriste se kako bi se zajamčila dosljednost stranica indeksa i podataka za korisničke i sustavne objekte u memoriji. Stranice koje upravljaju alokacijama zaštićene su internim stranicama za dodjelu SGAM (eng. Shared Global Allocation Map), GAM (eng. Global Allocation Map) i PFS (eng. Page Free Space). Ova vrsta čekanja akumulira se dok su latchevi u ekskluzivnom načinu rada. Ako je ovo čekanje visoko glavni uzrok tog problema je sukob s PFS-om [14].

- **LATCH_***

Ova vrsta čekanja povezana su sa laganom kratkotrajnom sinkronizacijom objekata koji su korišteni za zaštitu pristupa internoj cache memoriji, ali ne i buffer cache memoriji. Ova čekanja mogu indicirati razne probleme ovisno o vrsti latch-a. Određivanjem određene latch klase koja je akumulirala najviše vremena čekanja može se provjeriti upitom sys.dm_dos_latch_stats DMV [14].

- **ASYNC_NETWORK_IO**

Ovo se čekanje pogrešno tumači sa bottleneck-om mreže. Ustavi najčešći uzrok ovog čekanja je aplikacija klijenta koja procesira podatke red-po-red. Ispravljanje ove vrste čekanja

tipično zahtjeva promjenu koda na strani klijenta kako bi se rezultat upita pročitao što je prije moguće, samim time brže izvršio [9].

- **OLEDB**

Pojavljuje se kada nit uputi poziv strukturi koja koristi OLEDB (eng. Object Linking and Embedding Database), davatelja usluga čeka dok davatelj usluga pošalje nazad podatke. Česti uzrok ovog problema su upiti na povezanim (eng. Linked) serverima. Međutim mnogi DMV-ovi koriste OLEDB interno, tako česti pozivi tim DMV-ovima mogu uzrokovati tu vrstu čekanja. DBCC (eng. Database Console Command) provjere također koriste OLEDB pa je uobičajeno vidjeti ovu vrstu čekanja kada se pokrenu takve provjere [8].

2.3. Sql savjeti eng. Hints

SQL server savjeti su instrukcije koje nadjačavaju automatske odabire SQL tuning advisora. Prema terminologiji riječ savjet je zavaravajuća zato što u stvarnosti prisiljava se promjena načina rada odabira prilikom pokretanja plana izvršavanja [15].

Kada se pokreće spremljena procedura ili parametrizirani upit, SQL server kreira vlastiti plan izvršavanja odmah prilikom pokretanja. Kada se drugi puta izvrši upit ili spremljena procedura ne kompajlira se ponovno nego se koristi plan upita koji se stvorio prije. Tim načinom ne moraju se svaki puta procedure ili upiti rekompajlirati.

Standardno ponašanje SQL servera uobičajeno poboljšava performanse, ali ako je distribucija podataka na tablicama nepravilna može uzrokovati sniffing parametara. Sniffing parametara predstavlja proces izvršavanja spremljenih procedura koje sadrže parametre. Kada se procedura kompajlira vrijednost koja se prenosi preko parametara se procjenjuje, a ta procjena se uzima u obzir prilikom kreiranja plana izvršavanja. Vrijednost je pohranjena u plan izvršavanja odnosno u njegov cache. U ponovnim izvršavanjima koristi se ista vrijednost i isti plan. Kako bi se riješio taj problem mogu se koristiti savjeti za ponovno sastavljanje upita (eng. Recompile Query Hints) kako bi se generirao plan svaki puta kada bi se upit izvršavao [15].

2.3.1. JOIN savjeti

JOIN savjeti govore SQL optimizatoru koju JOIN strategiju će koristiti za povezivanje dviju tablica. Ovi savjeti forsiraju optimizatora da koristi NESTED LOOPS, HASH MATCH ili MERGE algoritam.

Ako se za spajanje dviju tablica koristi jedna manja tablica koja ima manje od 10 redova, a druga tablica veća, te sadrži indekse na stupcima za spajanje tada će najbolji savjet biti LOOP JOIN. Razlog toga je što LOOP JOIN zahtjeva manje usporedbi [2].

Ako se za spajanje koriste dvije tablice srednjih veličina koje su sortirane po stupcu na kojem se spajaju tada se koristi MERGE JOIN. Ako su obje tablice veće ili približno jednake veličine tada se koristi MERGE JOIN sa sortiranjem i HASH JOIN. Ako se dvije tablice znatno razlikuju po veličini HASH JOIN je najbrža opcija. On efikasno procesira velike, ne sortirane i ne indeksirane tablice [2].

REMOTE omogućuje izvođenje spajanja tablice s udaljenom (eng. Remote) tablicom s lokalnim serverom. On je koristan kada je jedna tablica lokalna, a druga udaljena također on bi se trebao koristiti samo ako lokalna tablica ima manje redova nego udaljena. Ako su obje tablice

udaljene iz istog izvora podataka tada REMOTE nije potreban. Koristi se samo kod INNER JOIN operacije.

2.3.2. Tablični savjeti

Tablični savjeti koriste se kada se eksplicitno želi forsirati određena locking strategija, ili metoda pristupa za tablice. Ova vrsta savjeta ne modificira zadane parametre i primjenjuje se samo za vrijeme trajanja upita. Neki od savjeta su: FORCESEEK, FORCESCAN, NOLOCK, HOLDLOCK, TABLLOCK, ROWLOCK itd.

2.3.3. Savjeti upita

Savjeti upita utječu na cijeli set operatora upita ne na samo individualne dijelove. Savjeti upita mogu biti JOIN savjeti, tablični savjeti, ili pak nekoliko savjeta koji su relevantni za upit.

Neki uobičajeni savjeti uključuju OPTIMIZE FOR, RECOMPILE, FORCE ORDER, FAST <rows>. Također savjeti se mogu specificirati nakon upita pomoću WITH opcije [16].

2.3.4. Vodiči kroz plan

Vodiči kroz plan pružaju sličnu funkcionalnost kao savjeti upita u smislu da dopuštaju eksplicitnu kontrolu korisnika da upravlja s planom kojeg je optimizator upita odabrao. Vodiči mogu biti korišteni kao savjeti upita, mogu biti kompletno fiksirani te mogu biti unaprijed generirani. Glavna razlika između vodiča kroz plan i savjeta nad upitima jest njihova pridruženost upitima [17].

Dok savjeti upita ili tablični savjeti moraju biti eksplicitno navedeni unutar upita oni nisu izbor ako ne postoji kontrola oko izvornog koda koji generira te upite. Ako aplikacija koristi ad-hoc upite umjesto spremljenih procedura, pogleda i funkcija, jedini način da se upravlja planom upita jest pomoću vodiča kroz plan. Oni se najčešće koriste za ublažavanje problema performansi kod programa treće strane.

Vodič kroz plan sastoji se od činjenica da plan pokretanja mora biti prilagođen ili kroz OPTION klauzulu koja ispisuje željene savjete za upite ili kroz potpuni XML plan upita koji se primjenjuje sve dok je on valjan [17].

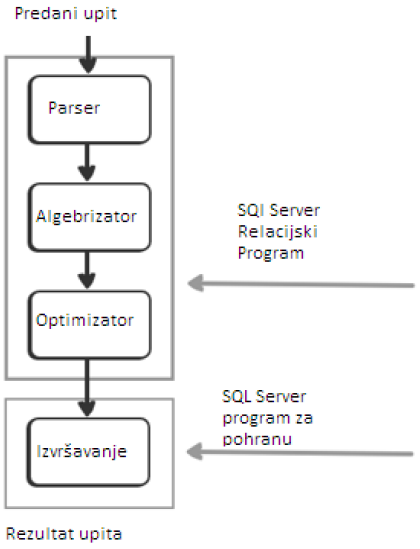
SQL Server podržava tri vrste vodiča kroz plan:

- Plan objekta vodi ciljne izjave koje se izvode unutar opsega koda objekta kao što su spremljene procedure, funkcije ili okidači (eng. Triggers). Ako se ista izjava nalazi u drugom kontekstu vodič za plan se ne primjenjuje.
- SQL vodič kroz plan koristi se za povezivanje općih ad-hoc izjava koje nisu unutar opsega koda objekta.
- Plan predloška može se koristiti pri apstraktnoj izjavi predloška koja se razlikuje samo u vrijednostima parametara. Mogu se koristiti za nadjačavanje postavke PARAMETRIZATION prilikom korištenja obitelji upita (upiti roditelj-djeca) [18].

3. PLAN IZVRŠAVANJA U SQL-u

Podnošenjem T-SQL upita govorimo SQL Server programu što želimo bez specificiranja kako da on taj upit izvrši. Između podnošenja upita i vraćanja rezultata upita krajnjem korisniku SQL Server program izvršiti će četiri Internet operacije kako bi konvertirao upit u format kojeg koristi SQL Server Storage preogram kako bi dohvatio određene podatke korištenjem procesa dodijeljenih SQL programu iz operacijskog sustava za rad na poslanom upitu [19].

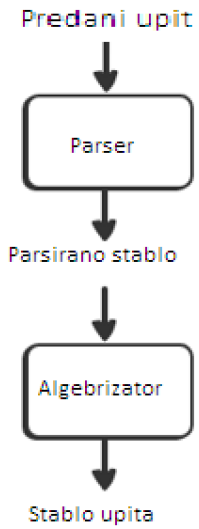
Prva tri procesa prilikom izvršavanja SQL Server Relation programa su: Parsiranje, algebrizacija te optimizacija. S druge strane korak izvršenja izvodi SQL Server Storage program. Ako podneseni upit nije DML (eng. Dana Manipulation Language) kao npr. CREATE TABLE ili ALTER TABLE nema potrebe za optimizacijom takvog upita zato što SQL Server program ima samo jedan način za izvođenje tih naredbi [20]. Četiri koraka procesiranja upita SQL Server Relational programa i SQL Server Storage programa prikazani su na slici 3.1.



Slika 3.1: Četiri koraka procesiranja upita

Kada je upit predan, on se dijeli u nekoliko individualnih dijelova, vrši se provjera nad tim dijelovima od strane SQL Server Relatioinal programa kako bi se potvrdilo da su ispravno napisani bez grešaka u sintaksi npr. proces koji se naziva parsiranje upita (eng. Query parsing). Rezultat parsiranja upita je parsirano stablo. Parsirano stablo interni je prikaz upita koji uključuje sve korake, također poznate kao preventivne operacije nakon kojih slijedi izvršavanje navedenog upita. Nakon stvaranja parsiranog stabla za DML upite, algebrizator će uzeti generirano stablo i razriješiti nazive različitih objekata u bazi podataka na koje se upućuje poslani upit u odnosu na sistemski katalog (eng. System catalog), kako bi se uvjerio da ti objekti u bazi podataka postoje,

te da korisnik koji je poslao upit ima prava za izvršavanje tog upita [19]. Algebrizator će generirati stablo upita koje će se koristiti u sljedećem koraku vidljivom na slici 3.2.



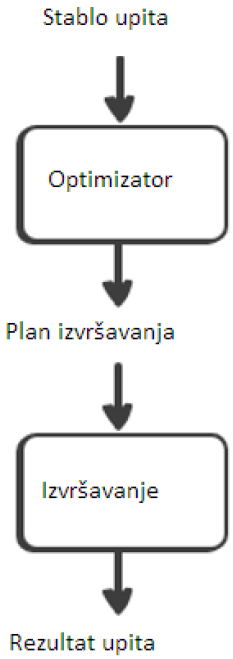
Slika 3.2: Stablo generirano od strane algebrizatora

Rezultat upita stabla procesora koji je predan od strane algebrizatora biti će unesen u optimizator upita (eng. Query optimizer) za plan izvršavanja SQL Servera. Optimizator upita funkcijski je odgovoran za modeliranje načina na koji SQL Server Relational program radi. On to čini kreiranjem najefikasnijeg plana izvršavanja za dani upit, uz najmanju potrošnju resursa SQL Servera. Takav plan naziva se plan izvršavanja baziran na troškovima. Optimizator će također isprobati više metoda izvršavanja, čitati će sve redove tablica koristeći različite indekse, različite JOIN-ove te različite redoslijede kako bi pronašao najoptimalniji plan izvršavanja sa što manje troška. Ukupni trošak svakog plana računa se dodjeljivanjem težine za svaku operaciju, zatim se sumiraju troškovi svih operacija korištenih u upitu [20].

Plan izvršavanja SQL Servera predstavlja binarnu reprezentaciju koraka koje prati SQL Server program kako bi izvršio upit. Drugim riječima najefikasniji i najjeftiniji put generiran od strane optimizatora upita koji prati različite algoritme prilikom izvršavanja danog upita. Plan se kreira na temelju upita procesnog stabla čiji su rezultat tablice baza podataka te indeksi statistike koji opisuju distribuciju i jedinstvenost podataka u objektima baze podataka. Navedena statistika pomaže optimizatoru upita da usporedi broj redove koja dohvaća skeniranjem cijele tablice i svih redova te broj redova dobivenih korištenjem različitih indeksa. U oba slučaja optimizator također ima na uvid trošak svake operacije [19]. Jako je važno uvijek statistiku imati ažuriranu kako bi se uvijek mogao kreirati najoptimalniji plan izvršavanja.

Kada se odabere najefikasniji plan izvršavanja predanog upita, tada se on sprema u plan cache memoriju. Plan cache memorija je dio SQL Server buffer-a gdje se planovi upita spremaju te se mogu u budućnosti ponovno koristiti, što je puno efikasnije nego svaki puta generirati novi plan. Sprema se zbog činjenice da je generiranje najoptimalnijeg plana skup proces. Kada se primi novi upit optimizator će prvo pretražiti plan cache memoriju kako bi se ponovno iskoristio već postojeći plan. Ako optimizator ne može pronaći plan, tada se vraća u prijašnje procese te kreira novi plan, a samim time uzima više vremena za samo izvršavanje upita. Ponovno korištenje planova može biti korisno za spremljene procedure koje se često izvršavaju s različitim ulaznim parametrima, a da pritom koriste isti plan izvršavanja. Kada imamo veliki broj ad-hoc upita, baza podataka spriječiti će ponovno korištenje planova i zahtijevat će konstanto generiranje novih [20].

U specifičnim situacijama optimizator preferirat će generiranje osnovnog plana takozvanog trivijalnog plana, kako bi izvršio upit koji nema agregacija ili kompleksne kalkulacije. Samim time optimizator ubrzava vrijeme izvršavanja te troši manje resursa pri generiranju efektivnog plana za upit [20]. Nakon toga plan izvršavanja biti će korišten od strane SQL Server Storage program, te će izvršiti zatražene procese poput dohvaćanja podataka, dodavanje novih ili modificiranje postojećih kao što je prikazano na slici 3.3.:



Slika 3.3: Procesi dohvaćanja podataka

SQL Server administrator baze podataka često ovisi o planovima izvršavanja odnosno popravljanja loše optimiziranih upita i lociranja problema prilikom izvršavanja upita. Plan izvršavanja administratoru daje odgovor na pitanja zašto je upit spor, zašto upit konzumira veliki udio procesora memorije i drugih resursa te zašto se ne koristi indeks. Također plan izvršavanja pomaže administratoru kako bi ponovno napisao upite kako bi brže radili odnosno da rade na što efikasniji način. Uz pomoć izvršenja upita administrator isto tako mora posjedovati određena znanja i vještine kako bi kreirao efikasan plan, odnosno administrator mora biti sposoban analizirati komponente plana [19].

3.1. Plan izvršavanja unutar SQL server management studija

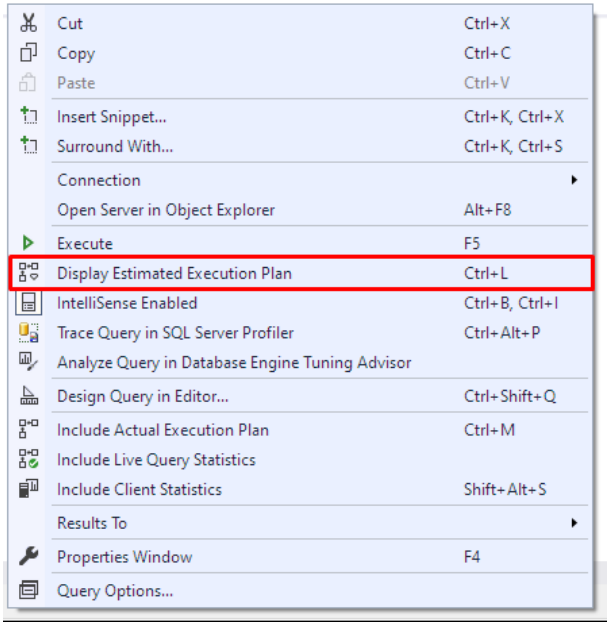
U Microsoft SQL Server Management Studio-u plan izvršavanja predstavlja grafički prikaz različitih koraka koji se odvijaju prilikom dohvaćanja rezultata upita iz baze podataka. Kada se upit počne izvršavati, program za samo procesiranje vrlo brzo generira nekoliko planova i bira onog najboljeg i najbržeg tj. najoptimalnijeg. Postoje dvije vrste planova, a to su:

Procijenjeni plan izvršavanja kao što samo ime nalaže ovaj tip plana izvršavanje je samo aproksimacija od procesora upita. On procjenjuje koji specifični koraci će biti uključeni za vrijeme vračanja rezultata upita. On se inače generira odmah prije nego se upit kreće izvršavati.

Stvarni plan izvršavanja generira se nakon izvršavanja upita. Pokazuje stvarne operacije i korake koji su se izvodili za vrijeme izvršavanja upita. Može se razlikovati od procijenjenog plana, ali i ne mora.

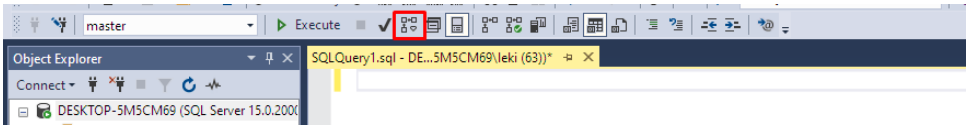
3.2. Prikazivanje procijenjenog plana izvršavanja

Kada se upit u potpunosti napiše, kombinacijom tipki „*Ctrl + L*“ može se generirati procijenjeni plan izvršavanja. Isto tako procijenjeni plan može se generirati pritiskom desnog klika na prozor od upita, zatim se odabere opcija „*DISPLAY ESTIMATED EXECUTION PLAN*“ iz menija, kao što je prikazano na slici 3.4.



Slika 3.4: Odabir opcije „*DISPLAY ESTIMATED EXECUTION PLAN*“

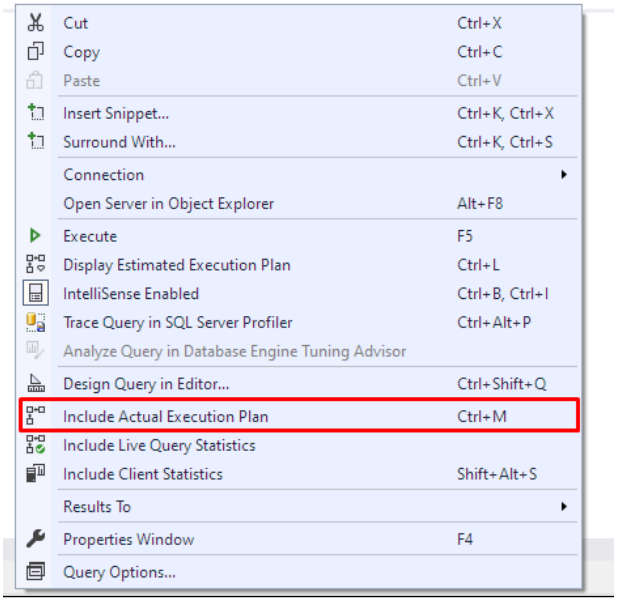
Postoji još jedan alternativni način prikaza procijenjenog plana, a to je odabirom gumba na alatnoj traci kao što je prikazano na slici 3.5.



Slika 3.5: Alternativni način prikaza procijenjenog plana

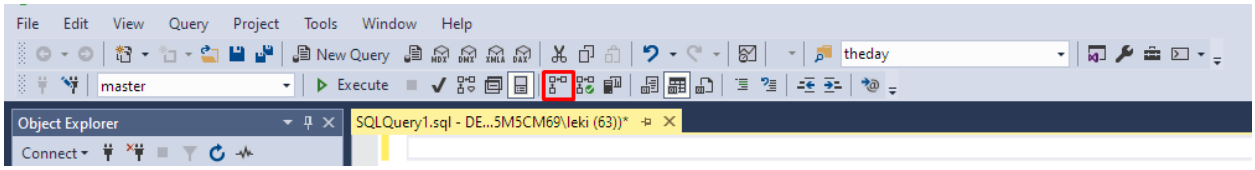
3.3. Prikazivanje stvarnog plana izvršavanja

Kombinacijom tipki „*Ctrl + M*“ možemo generirati stvarni plan izvršavanja nakon što se upit uspješno izvršio. Isto tako toj opciji možemo pristupiti desnim klikom na prozor upita i odabirom opcije „*INCLUDE ACTUAL EXECUTION PLAN*“ kao što je prikazano na slici 3.6.



Slika 3.6: Odabir opcije „*INCLUDE ACTUAL EXECUTION PLAN*“

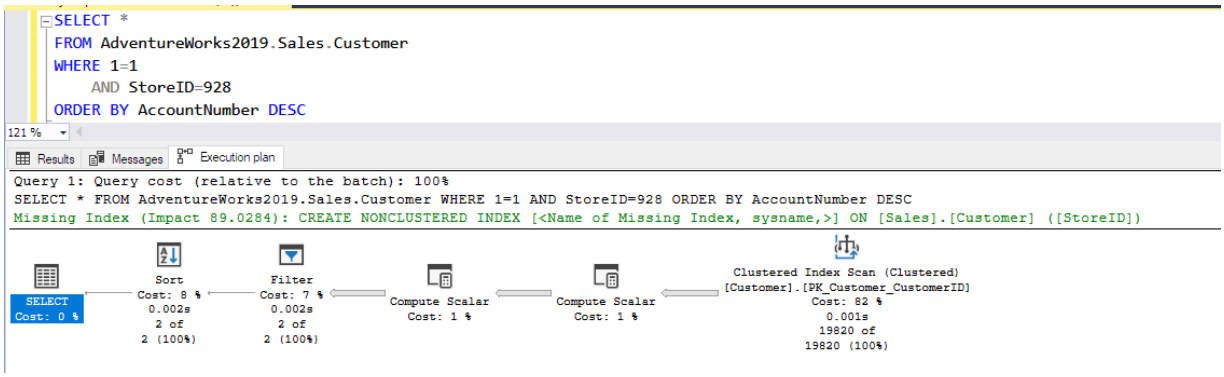
Također alternativna metoda prikazivanja stvarnog plana je pritiskom ikone na alatnoj traci prikazanoj na slici 3.7.



Slika 3.7: Alternativni način prikaza stvarnog plana

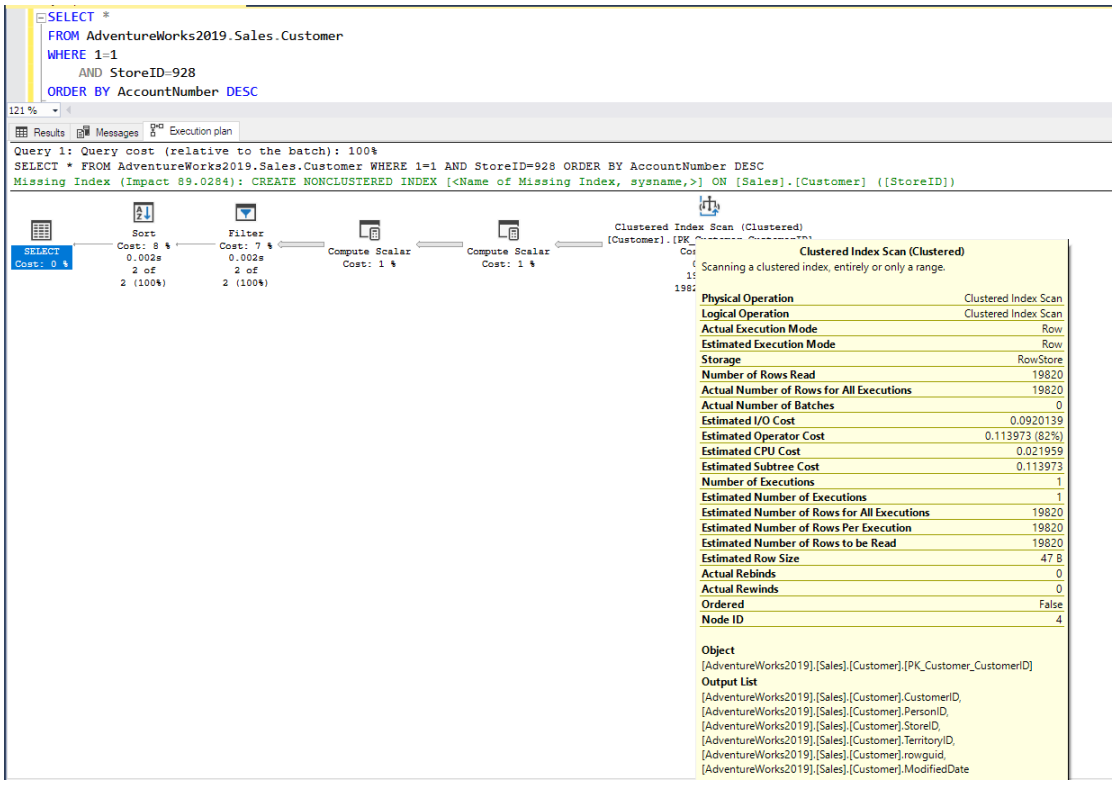
3.4. Interpretacija plana

Nakon što se plan generira, u prozoru upita pojavit će se pored kartice „*Results*“ i „*Messages*“ kartica „*Execution Plan*“ kao što je vidljivo na slici 5.8. Za jednostavne upite poput ovog na slici 3.8. procijenjeni plan biti će jednak stvarnom planu.



Slika 3.8: Prikaz plana izvršavanja

U planu izvršavanja kada kursorom pređemo preko komponenti možemo vidjeti detaljnije statistike svake operacije i svake komponente kao na slici 3.9. Plan se interpretira u smjeru desna na lijevo te u smjeru od gore prema dolje. Kada se plan nalazi u jednom redu nema potrebe ići od gore prema dolje.

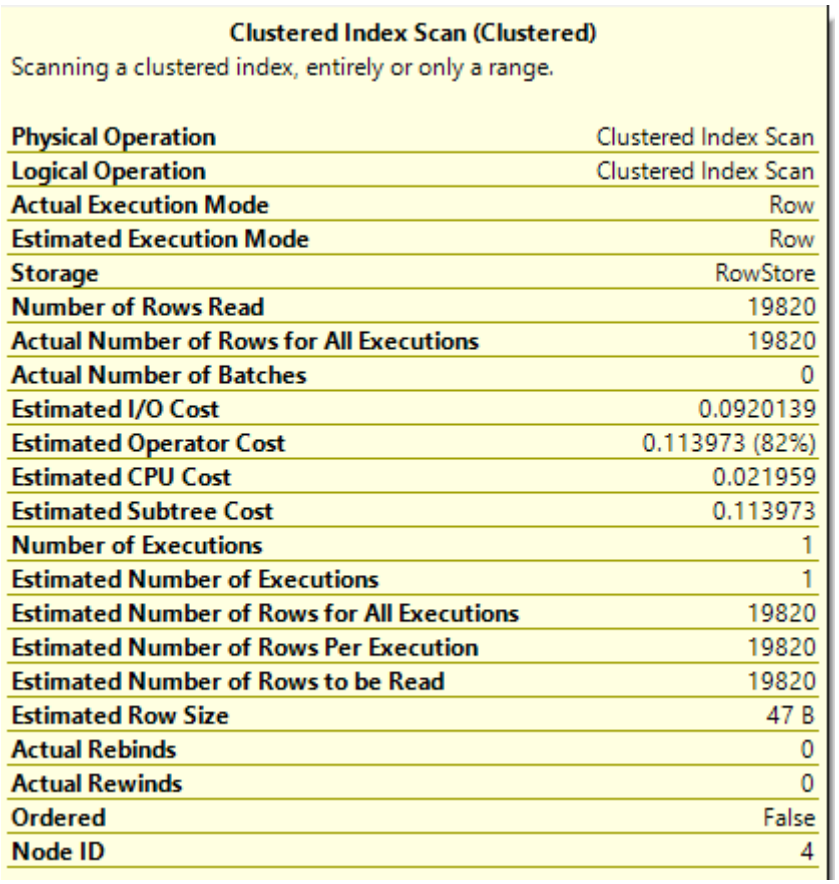


Slika 3.9: Detaljnije statistike svake operacije

Plan izvršavanja možemo podijeliti na sljedećih pet koraka: Skeniranje klasteriranog indeksa (eng. Clustered Index Scan), Tok podataka iz skeniranja klasteriranog indeksa (eng. Data Flow from Clustered Index Scan), Operator sortiranja (eng. Sort Operator), Tok podataka iz operatora sortiranja (eng. Data Flow from Sort Operator) i Operator odabira (eng. Select Operator)

3.5. Skeniranje klasteriranog indeksa

Prva komponenta kada pratimo plan s desna na lijevo je skeniranje klasteriranog indeksa. On se izvršava na primarnom ključu varijable [21]. Ostatak komponenti prikazanih skeniranog klasteriranog indeksa prikazan je na slici 3.10.



Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	19820
Actual Number of Rows for All Executions	19820
Actual Number of Batches	0
Estimated I/O Cost	0.0920139
Estimated Operator Cost	0.113973 (82%)
Estimated CPU Cost	0.021959
Estimated Subtree Cost	0.113973
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	19820
Estimated Number of Rows Per Execution	19820
Estimated Number of Rows to be Read	19820
Estimated Row Size	47 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	4

Slika 3.10: Prikaz svih komponenti klasteriranog indeksa

- Fizička operacija (eng. Physical Operation) predstavlja vrstu operacija koja je implementirana od strane uputa logičkih operatora. Svi fizički operatori predstavljaju objekte koji izvode neku operaciju, npr. skeniranje klasteriranog indeksa, traženje indeksa itd.
- Logička operacija (eng. Logical Operation) ova vrsta operacija opisuje stvarnu matematičku operaciju koja je korištena prilikom izvršavanja upita. Neki od primjera su: RIGHT ANTI SEMI JOIN, SEGMENT REPARTITION itd.
- Stvarni način izvršavanja (eng. Actual Execution Mode) predstavlja stvarni način izvršavanja korišten od strane procesnog programa kako bi se izvršio upit. Primjer je ROW i BATCH.
- Procijenjeni način izvršavanja je vrlo sličan stvarnom načinu, ali pokazuje procijenjene vrijednosti.

- Pohrana (eng. Storage) nam govori na koji način će optimizator spremiti rezultate nakon što ih izvuče iz upita.
- Broj redova čitanja (eng. Number of Rows Read) vraća ukupni broj zapisa koji su pročitani iz tablice indeksa.
- Stvarni broj redova (eng Actual Numbers of Rows) nam govori ukupni broj svih zapisa koji su dohvaćeni iz upita pomoću WHERE klauzule.
- Stvarni broj serija (eng. Actual Number of Batches) ako je način izvršavanja BATCH zatim će se vratiti lista serija koje su izvršene kako bi se dohvatio rezultat upita.
- Procijenjeni I/O trošak (eng. Estimated I/O Cost) govori nam koliko je ulaznih/izlaznih operacija u rezultatu
- Procijenjeni trošak operatora (eng. Estimated Operator Cost) ne predstavlja točan iznos troška nego relativne informacije.
- Procijenjeni trošak procesora (eng. Estimated CPU Cost) predstavlja cijenu procesora koja će se dogoditi kako bi se izvršila operacija.
- Procijenjeni trošak pod stabla (eng. Estimated Subtree Cost) predstavlja trošak stabla izvršavanja koje se trenutno čita s desna na lijevo i od gore prema dolje.
- Broj izvršavanja (eng. Number of Executions) govori nam broj izvršavanja s kojima je optimizator morao baratati u jednoj seriji.
- Procijenjeni broj izvršavanja (eng. Estimated Number of Executions) sličan broju izvršavanja samo procijenjena vrijednost.
- Procijenjeni broj redova (eng. Estimated Number of Rows) broj redova koje optimizator procjenjuje da će biti vraćeni.
- Procijenjeni broj redova za čitanje (eng. Estimated Number of Rows to be Read) broj redova koje optimizator misli da će biti pročitani.
- Procijenjeni broj veličine retka (eng. Estimated Row Size) veličina pohrane svakog reda.
- Stvarna ponovna povezivanja (eng. Actual Rebinds) govori nam koliko puta se dogodila ponovna evaluacija objekta kako bi se izvršio proces upita.

- Stvarna premotavanja (eng. Actual Rewinds) nam govori da li je bilo promjena u koreliran-im vrijednostima za objekt koji se obrađuje.
- Poredanost (eng. Ordered) svojstvo koje govori da li je set podataka na kojem se operacija izvodi u sortiranom stanju ili ne.
- ID čvora (eng. Node ID) predstavlja automatsku dodjelu broja redoslijeda kojim je operator pozvan u planu izvršavanja.

4. KAKO POBOLJŠATI UPITE

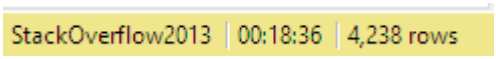
Baza podataka korištena u ovom poglavlju je Stack Overflow [25]. Korištena je Medium: 50GB baza podataka od 2013. godine. Baza prilikom skidanja ima 10GB, raspakiranjem zipa pretvara se u 50GB. Korištena je navedena baza zbog upita koji su sporiji da budu spori, zato što u bazi od 10GB spori upiti se brzo izvršavaju, a cilj ovog dijela je vidjeti razliku vremena izvršavanja.

4.1. Prvi upit

Za prvi primjer odabran je upit sa izvora [22] koji je vidljiv na slici 4.1. Kako bi prilagodili datume postavili smo datume da odgovaraju bazi podataka iz 2013. godine. Izvođenje upita trajalo je 16 minuta i 26 sekundi sa ostvarenim rezultatima od 4238 redova što je vidljivo na slici 4.2.

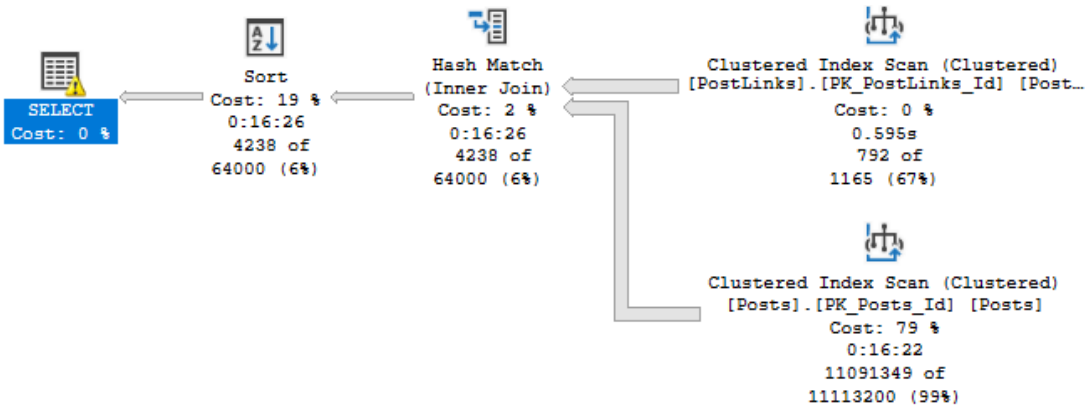
```
SELECT
      PostLinks.CreationDate
    , Posts.*
FROM StackOverflow2013.dbo.PostLinks PostLinks
INNER JOIN StackOverflow2013.dbo.Posts Posts ON Posts.ParentId=PostLinks.RelatedPostId
WHERE 1=1
    AND PostLinks.LinkTypeId = 3
    AND PostLinks.CreationDate > '2013-09-13'
    AND PostLinks.CreationDate < '2013-09-20'
    AND Posts.PostTypeId = 2
ORDER BY PostLinks.CreationDate DESC;
```

Slika 4.1: Prvi upit



Slika 4.2: Vrijeme izvršavanja prvog upita

Iz plana izvršavanja na slici 4.3. vidljivo je da ima nekoliko klasteri-ranih skeniranja indeksa, kao i INNER JOIN te sortiranje rezultata.



Slika 4.3: Plan izvršavanja prvog upita

Također može se primijetiti žuti uskličnik na SELECT-u. Prolaskom miša možemo vidjeti upozorenje za memoriju koje je vidljivo na slici 4.4.

SELECT

Cached plan size	96 KB
Estimated Operator Cost	0 (0%)
Degree of Parallelism	1
Estimated Subtree Cost	3920.97
Memory Grant	374 MB
Estimated Number of Rows for All Executions	0
Estimated Number of Rows Per Execution	63999.6

Statement

```
SELECT
  PostLinks.CreationDate
, Posts.*
FROM StackOverflow2013.dbo.PostLinks PostLinks
INNER JOIN StackOverflow2013.dbo.Posts Posts ON
Posts.ParentId=PostLinks.RelatedPostId
WHERE 1=1
AND PostLinks.LinkTypeId = 3
AND PostLinks.CreationDate > '2013-09-13'
AND PostLinks.CreationDate < '2013-09-20'
AND Posts.PostTypeId = 2
ORDER BY PostLinks.CreationDate DESC
```

Warnings

The query memory grant detected "ExcessiveGrant", which may impact the reliability. Grant size: Initial 382696 KB, Final 382696 KB, Used 9456 KB.

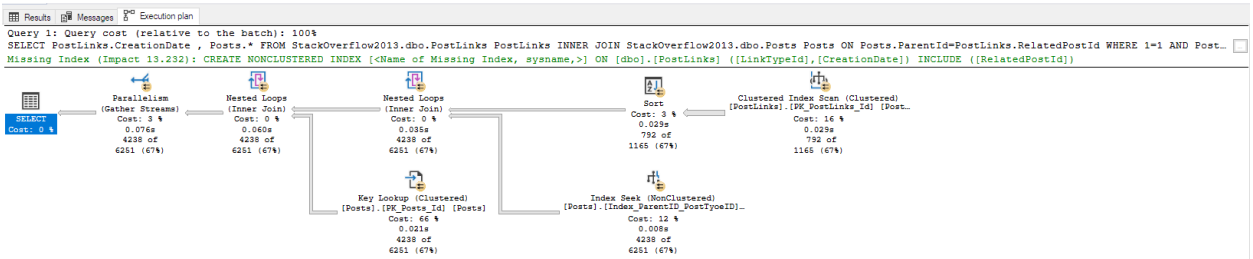
Slika 4.4: Upozorenje za memoriju za prvi upit

Za početak poboljšanja vremena izvršavanja korisno bi bilo kreirati INDEX na stupce na kojima radimo JOIN. Kôd je vidljiv na slici 4.5. Vrijeme trajanja kreiranja indeksa iznosilo je 9 minuta i 5 sekundi.

```
CREATE INDEX Index_ParentID_PostTyoeID
ON dbo.Posts(ParentID, PostTypeId)
```

Slika 4.5: Kôd za kreiranje indeksa

Nakon kreiranja indeksa ponovno je pokrenut upit. Rezultati upita vidljivi su na slici 4.6.



Slika 4.6: Rezultati prvog upita

Vrijeme trajanja izvršavanja iznosilo je 1 sekundu. Također u SQL planu izvršavanja imamo zelenim fontom preporuku optimizatora da nam nedostaje indeks odnosno preporučuje nam da kreiramo neklasterirani indeks na bazi PostLinks. Zato što je upit izvršen u sekundi nije potrebno dodatno tuniranje. Isto tako može se primijetiti da na SELECT kućici nema više žutog uskličnika za upozorenje za memoriju. Također nema SORT kućice i svi SCAN-ovi u upitu postali su SEEK-ovi.

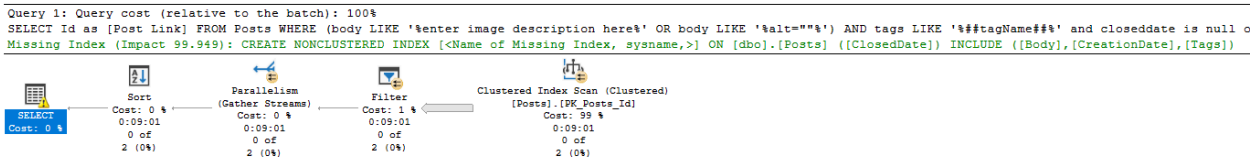
4.2. Drugi upit

Za drugi primjer odabran je upit sa sljedećeg izvora [23]. Cilj ovog upita je pronaći postove sa slikama koje imaju opis *„enter image description here“*. Kôd je vidljiv na slici 4.7.

```
SELECT Id as [Post Link]
FROM Posts
WHERE (body LIKE '%enter image description here%' OR body LIKE '%alt=""%')
  AND tags LIKE '%##tagName##%'
and closeddate is null
order by creationdate desc
```

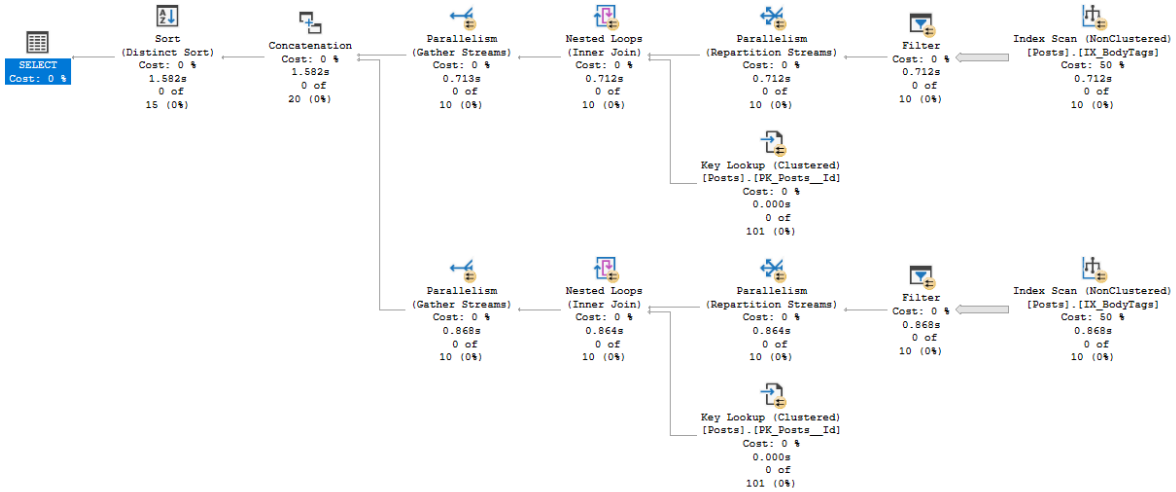
Slika 4.7: Kôd drugog upita

Na slici 4.8. vidljivo je da je trajanje upita 9 minuta i 1 sekunda, također vidljivo je da je potreban indeks. Potrebno je kreirati ne klasterirani indeks na bazi Posts. Bitno je napomenuti da upit ima 4201440 logičkih čitanja.



Slika 4.8: Trajanje drugog upita

Nakon primjenjivanja indeksa i promjene strukture upita, on se izvršava za 1.372 sekundi slika 4.9.



Slika 4.9: Rezultat drugog upita

Kôd je vidljiv na slici 4.10.

```
SELECT PostLink=Id
FROM Posts
WHERE 1=1
    AND Body LIKE '%enter image description here%'
    AND tags LIKE '%##tagName##%'
    AND ClosedDate IS NULL
UNION
SELECT PostLink=Id
FROM Posts
WHERE 1=1
    AND Body LIKE '%alt=""%'
    AND tags LIKE '%##tagName##%'
    AND ClosedDate IS NULL
OPTION (RECOMPILE)
```

Slika 4.10: Kôd drugog upita

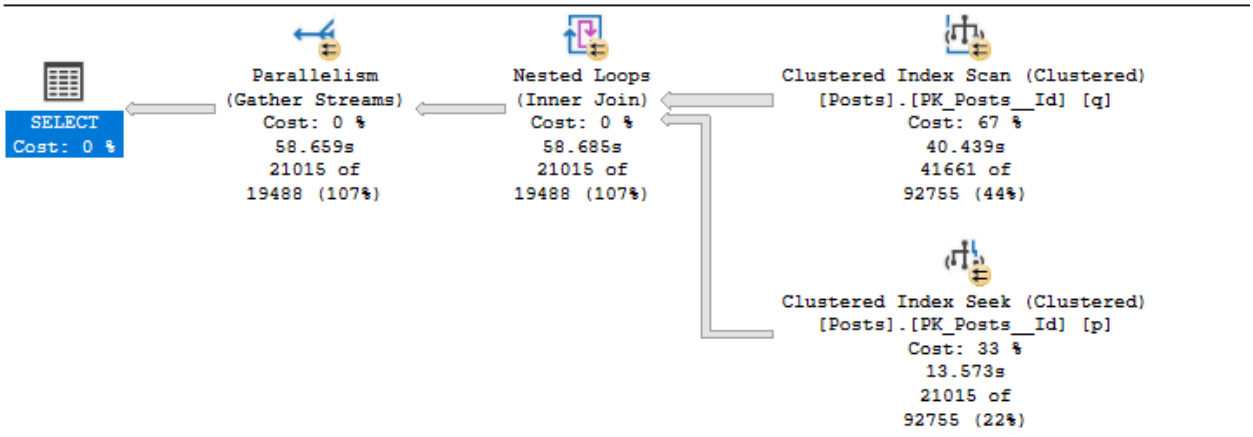
4.3. Treći upit

Treći upit odabran je sa izvora [24]. Njegovo vrijeme izvršavanja je 58 sekundi, ima logičkih čitanja 987759 te fizičkih čitanja 75. Kôd se može vidjeti na slici 4.11.

```
select
 p.OwnerUserId as [User Link],
 p.Id as [Post Link],
 p.Score
from Posts p
inner join Posts q on q.Id = p.ParentID
where p.PostTypeId = 2 and p.Score > 5
and q.Score > 3 and q.AnswerCount = 1
and q.AcceptedAnswerId = p.Id
```

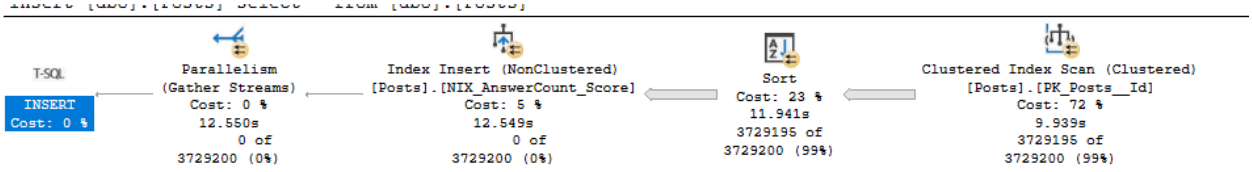
Slika 4.11: Kod trećeg upita

Nakon izvršenja upita na planu je vidljivo da je korišten jedno skeniranje indeksa, i jedno pretraživanje indeksa. Kroz INNER JOIN korištena je ugniježđena petlja, te su se podaci preko paralelizma dohvaćali za konačni prikaz kao što je vidljivo na slici 4.12.



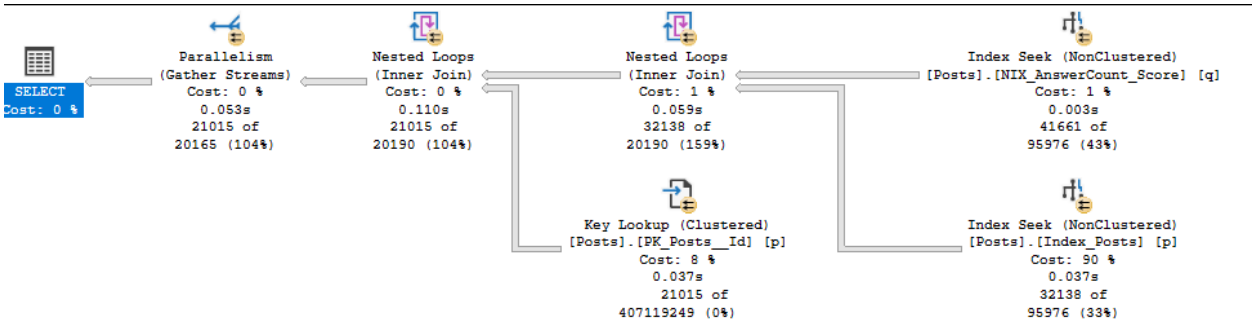
Slika 4.12: Plan izvršavanja trećeg upita

Kreiranje indeksa nad prvom tablicom trajalo je oko 12 sekundi, plan izvršavanja vidljiv je na slici 4.13.



Slika 4.13: Kreiranje indeksa za treći upit

Nakon kreiranja indeksa, upit je ponovno pokrenut te je njegovo izvršavanje trajalo jednu sekundu, kao što je vidljivo na slici 4.14.



Slika 4.14: Rezultat trećeg upita

5. ZAKLJUČAK

U ovome radu objašnjen je pojam indeksiranja. Prikazana je arhitektura indeksa, kao i glavni primjeri poput ne klasteriranog, klasteriranog, filtriranog i Columnstore indeksa. Kako bi se povećala performansa upita indeksi ne bi smjeli biti rađeni samo za jedan upit nego oni moraju biti kompatibilni sa više upita kako bi što manje zauzimali memoriju. Skeniranje je skup proces, te kada se upiti izvršavaju uvijek glavni fokus treba biti na traženju (eng. Seek). U praksi se preporuča da ključevi klasteriranog indeksa moraju biti što kraći. Ne klasterirani indeks bi trebalo postaviti na svaki stupac koji se koristi za pretraživanje ili sortiranje. Columnstore indeksi pohranjuju sve podatke u različite stupce na različite stranice. To potpomaže I/O performansama, te efikasnijem iskorištenju memorije. On se primjenjuje u skladištu podataka kod tablica činjenica (eng. Fact tables).

Kada korisnik podnosi zahtjev SQL Serveru za dohvaćanje podataka, glavni element SQL Servera je njegovo vrijeme reagiranja odnosno vrijeme procesiranja upita. Svaki put kada je zahtjev stavljen na čekanje SQL Server sprema podatke o čekanju. Ti podaci se nazivaju statistika čekanja. Postoje dva dinamična pogleda kojim se oni mogu pristupiti, a to su sys.dm_os_wait_stats - agregirana statistika za sve vrste čekanja te sys.dm_os_waiting_tasks - statistika čekanja za trenutne zahtjeve

Potreba za optimizacijom upita vrlo je važna, ponajviše kod ugniježđenih upita koji su najkompleksniji i uzimaju najviše resursa. Takvi upiti poboljšavaju se korištenjem savjeta. Glavni tipovi savjeta su: JOIN, tablični savjeti i savjeti upita. Svaki ima određenu primjenu, ali sam optimizator odabire unaprijed najbolje savjete kako bi se iskoristilo što manje resursa, a upit izveo što brže.

Kako bi otkrili problem za spore upite, možemo koristiti alat koji je integriran u samom SQL Server Management studiju, a to je plan izvršavanja. On nam govori kojim redom, te koliko dugo su se koji procesi izvršavali kako bi se dohvatili željeni podaci. Također on nam daje preporuku ukoliko nam nedostaje neki indeks kako bi povećali performanse. Pomoću plana izvršavanja možemo vidjeti na koristi li se za dohvaćanje podataka skeniranje ili traženje.

POPIS SLIKA

LITERATURA

WEB-STRANICE

[1] S.F., Staff Contributor, Response Time Analysis: How to Improve Database Performance by Measuring User Experience, LogicalRead, dostupno na: https://logicalread.com/response-time-analysis/#.YmOksE5ByUk [10.06.2022.]

[2] Grant Fritchey - SQL Server Query Performance Tuning-Apress (2014) str. 111.-143.

[3] Peter Gulutzan, Trudy Pelzer - SQL Performance Tuning-Addison-Wesley Professional (2002)

[4] D.M, Madushan, How Database B-Tree Indexing Works, dostupno na: https://dzone.com/articles/database-btree-indexing-in-sqlite [10.06.2022.]

[5] A.Y, Yaseen, SQL Server index structure and concepts, SqlShack, dostupno na: https://www.sqlshack.com/sql-server-index-structure-and-concepts/ [10.06.2022.]

[6] S.A, Adhinkary, Introduction of B+ Tree, GeeksForGeeks, dostupno na: https://www.geeksforgeeks.org/introduction-of-b-tree/ [10.06.2022.]

[7] N.C., Crnko, Vrste indeksa prema internoj strukturi, srce.hr, dostupno na: https://sistemac.srce.hr/node/125[11.06.2022.]

[8] Benjamin Nevarez (auth.) - High Performance SQL Server_ The Go Faster Book-Apress (2016) str. 156-180

[9] SQL Server Performance Tuning Using Wait Statistics: A Beginner's Guide By Jonathan Kehayias and Erin Stellato str. 8-40.

[10]E.E., Erkec, Introduction to SQL Server Filtered Indexes, SQL Shack, dostupno na: https://www.sqlshack.com/introduction-to-sql-server-filtered-indexes/[10.06.2022.]

[11] Columnstore indexes: Overview, Microsoft, dostupno na: https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-overview?view=sql-server-ver16 [01.07.2022.]

[12] Columnstore index in sql server , SQLTREEO, dostupno na: https://www.sqltreeo.com/docs/columnstore-index-in-sql-server [01.07.2022.]

[13] E.E, Erkec, Boost SQL Server Performance with Wait Statistics , SQL Shack, dostupno na: https://www.sqlshack.com/boost-sql-server-performance-with-wait-statistics/ [01.07.2022.]

[14] SpotLight Cloud, dostupno na: https://app.spotlightcloud.io/waitopedia/waits [01.07.2022.]

[15] B.O., Ozar, The Elephant and the Mouse, or, Parameter Sniffing in SQL Server, BrentOzar, dostupno na: https://www.brentozar.com/archive/2013/06/the-elephant-and-the-mouse-or-parameter-sniffing-in-sql-server/ [01.07.2022.]

[16] Hints (Transact-SQL) – Query, Microsoft, dostupno na: https://docs.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-query?view=sql-server-ver16 [02.07.2022.]

[17] Plan Guides, Microsoft, dostupno na: https://docs.microsoft.com/en-us/sql/relational-databases/performance/plan-guides?view=sql-server-ver16[02.07.2022.]

[18] A.P, Prato, How to use a SQL Server Plan Guide to Tune Queries, MSSQLTips, dopstuno na: https://www.mssqltips.com/sqlservertip/1630/how-to-use-a-sql-server-plan-guide-to-tune-queries/[02.07.2022.]

[19] A.D., Das, Execution Plans in SQL Server, SQLShack, dostupno na: https://www.sqlshack.com/execution-plans-in-sql-server/ [05.07.2022.]

[20] E.E, Erkec, Explore the secrets of SQL Server execution plans, SQLShack, dostupno na: https://www.sqlshack.com/explore-the-secrets-of-sql-server-execution-plans/ [05.07.2022.]

[21] A.Y., Yaseen, How to Analyze SQL Execution Plan Graphical Components, SQLShack, dostupno na: https://www.sqlshack.com/how-to-analyze-sql-execution-plan-graphical-components/ [05.07.2022.]

[22] Select all master answers in one week, StackExchange, dostupno na: https://data.stackexchange.com/stackoverflow/query/1360210/select-all-master-answers-in-one-week [05.07.2022.]

[23] Top 15k posts with images that need a description (have "*enter image description here*"), StackExchange, dostupno na: https://data.stackexchange.com/stackoverflow/query/1613280/top-15k-posts-with-images-that-need-a-description-have-enter-image-description [05.07.2022.]

[24] Highlander badge, StackExchange, dostupno na: https://data.stackexchange.com/stackoverflow/query/8730/highlander-badge [05.07.2022.]

[25] B.O., Ozar, How to Download the Stack Overflow Database, BrentOzar, dostupno na: https://www.brentozar.com/archive/2015/10/how-to-download-the-stack-overflow-database-via-bittorrent/ [05.07.2022.]

SAŽETAK

U današnjem modernom svijetu i ubrzanom životu potrebno je što prije doći do željenih informacija, kako bi se povećala sama produktivnost. Upravo zbog toga postoji mnogo načina kako bi što prije došli do određenih informacija. Napredni načini za optimizaciju performansi u SQL programu nam omogućuju brži i efikasniji pristup željenim informacijama. U ovom radu opisana je arhitektura indeksa, vrste, te sam rad. Također je opisano ubrzanje zahtjeva pomoću statistike čekanja koja je implementirana u sam SQL Server. U radu se govori i o savjetima, odnosno njihovoj vrsti te kada ih primijeniti. Također opisan je plan izvršavanja te pojedine njegove pojedine komponente. Izvršeno je poboljšanje kroz tri upita. Početno izvršavanje trajalo je oko desetak minuta, poboljšanjem upita to vrijeme svedeno je na gotovo par sekundi.

Ključne riječi: Indeksi, Plan izvršavanja, Savjeti, SQL Server program, Ubrzavanje upita

ABSTRACT

In today's modern world and fast-paced life, it is necessary to get the desired information as soon as possible, in order to increase productivity itself. This is exactly why there are many ways to get certain information as soon as possible. Advanced Approaches to performance tuning in sql server engine allow us to access the desired information faster and more efficiently. This paper describes the architecture of the index, types, and how they work. It also describes how to speed up requests using the wait statistics implemented in SQL Server. The paper also talks about hints, i.e. their type and when to apply them. The execution plan and its individual components are also described. An improvement was made through three queries. The initial execution took about ten minutes, by improving the query, that time was reduced to a couple of seconds.

Key words: Execution plan, Hints, Indexes, SQL Server Engine, Query tuning

ŽIVOTOPIS

Leon Kupanovac rođen je 05.08.1995. godine u Osijeku u Republici Hrvatskoj. Odrastao je u Donjem Miholjcu, a trenutačno živi u Osijeku. Školovanje je započeo 2002. godine u Osnovnoj školi Ante Starčevića Viljevo (samo 1. razred osnovne škole), a ostatak osnovnoškolskog obrazovanja dovršio je u Osnovnoj školi Augusta Harambašića Donji Miholjac. Sve razrede osnovne škole završio je s odličnim uspjehom. Godine 2010. upisao je komercijalni smjer u Srednjoj školi Donji Miholjac. Završni rad bio mu je ocjenjen odličnim, a prosječni prolazak na maturi bio mu je vrlo dobar. Na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek (FERIT) upisao se 2014. na stručni smjer Informatike. 2018. Završio je tri godine stručnog studija, zatim je upisao razlikovne obveze. Nakon razlikovnih obaveza 2020. upisao je Diplomski sveučilišni studij Elektrotehnike, smjer komunikacije i informatike (izborni blok Mrežne tehnologije). Također završio je obuku za servisera osobnih računala i servera koja je trajala 155 sati 2016. godine u EDUNOVI školi informatike i managementa. 2020. godine osvojio je prvo mjesto u kuhanju piva na 9. Homebrew prvenstvu sa stilom Pale Ale. Ujedno je i član Panonskog Pivarskog Sindikata. 2021. te 2022. osvojio je srebrne medalje na sajmu inovacija INVENTUM u Iloku. Sudjelovao je u 18. konferenciji Data Saturdays Croatia 2022. godine. Trenutno radi u Addiko Banci u Osijeku kao specijalist za baze podataka u T-SQL okruženju.