**Univerzitet u Nišu**

**Elektronski fakultet**

MS SQL - Evaluacija operatora koji se koriste prilikom obrade upita

Predmet: Sistemi za upravljanje bazama podataka

Student: Profesor:

Nevena Čolić 1360 prof. dr. Aleksandar Stanimirović

# Sadržaj

[MS SQL - Opšte](#_5juropc86rgr) 3

[Obrada upita](#_ootc724d4ug6) 5

[MS SQL - Obrada upita](#_n543sowca8au) 9

[MS SQL - Optimizacija upita](#_kcua6rrc537e) 11

[Evaluacija operatora - Strategije](#_y9ho1uc8h3nh) 13

[Transact SQL - Evaluacija operatora](#_9mr4huui54fg) 16

[Literatura](#_5gyen0gcnhh6) 21

# MS SQL - Opšte

Microsoft SQL server jeste sistem za upravljanje relacionom bazom podataka koji podržava širok spektar obrade transakcija, poslovnu inteligenciju i analitiku u današnjim poslovnim IT okruženjima. Predstavlja jednu od tri vodeće tehnologije za upravljanje bazama podataka, zajedno sa Oraclom i IBM-om.

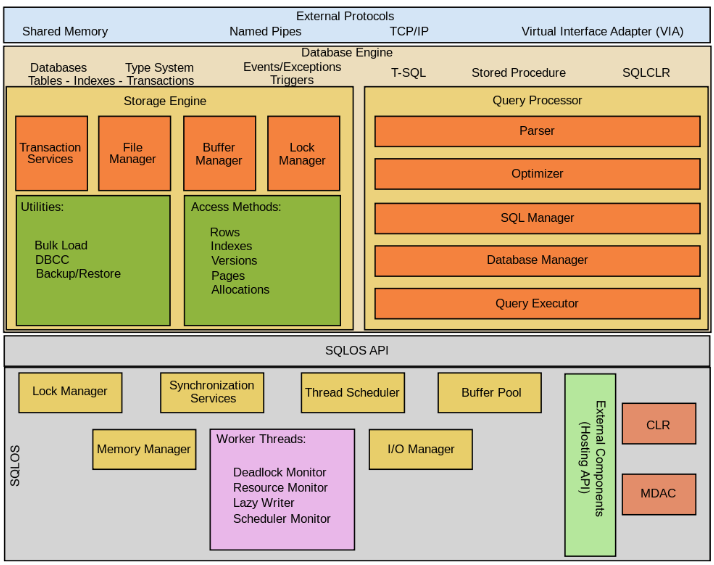
Izgrađen je preko SQL-a, standardnog jezika kod baza podataka i vezan je za Transact-SQL (T-SQL), Microsoftovu implementaciju jezika koja dodaje skup vlasničkih programskih produžetaka originalnom jeziku.

SQL Server je primarno izgrađen oko tabelarne strukture zasnovane na redovima, koja povezuje elemente iz različitih tabela izbegavajući potrebu da dodatno čuva iste podatke na nekoliko mesta. Relacioni model takođe pruža referentni integritet i druga ograničenja integriteta radi održavanja tačnosti podataka. Te provere su deo šireg poštovanja principa atomičnosti, konzistentnosti, izolovanosti i trajnosti što se zajedničkim imenom naziva ACID[[1]](#footnote-1) osobinama i dizajnirano je da garantuje pouzdanost baze podataka.

Sržna komponenta MS SQL-a jeste mehanizam baze podataka (eng. SQL Server Database Engine) koji kontroliše smeštanje podataka, obradu i bezbednost. Uključuje relacioni mehanizam koji obrađuje komande i upite kao i mehanizam za smeštanje fajlova, tabela, stranica, indeksa, nizova i transakcija. Objekti baze kao što su procedure, pogledi i specijalne procedure se takođe kreiraju i izvršavaju od strane mehanizma baze podataka (eng. Database Engine).

Relacioni mehanizam (eng. Relational Engine) sadrži komponente koje odlučuju koji je najbolji način za izvršavanje upita i on je poznat i pod nazivom procesor upita. Na osnovu upita, relacioni mehanizam zahteva podatke od mehanizma za smeštanje i zatim obrađuje rezultate. U neke od njegovih odgovornosti spadaju obrada upita, upravljanje memorijom, nitima i zadacima, upravljanje baferima i distribuirana obrada upita.

Ispod relacionog mehanizma i mehanizma za smeštanje podataka leži operativni sistem - SQL Server Operating System (eng. skr. SQLOS). SQLOS upravlja funkcijama nižeg nivoa, kao što je upravljanje memorijom i ulazom i izlazom, raspoređivanje taskova i zaključavanje podataka da bi se izbegli konflikti nad istim.



**Slika 1. MS SQL arhitektura**

Za upravljanje podacima, SQL Server uključuje različite servise kao što su servisi za kvalitet podataka, servisi za ispitivanja ali i master servisi: SQL Server Interogation Services (skr. SSIS), SQL Server Data Quality Services i SQL Server Master Data Services.

Iznad mehanizma baze jeste sloj koji čini mrežni interfejs koji koristi Microsoftov protokol za tabelarni tok podataka da bi upravljao zahtevima i odgovorima koji dolaze sa servera. Na poslednjem, korisničkom nivou, upiti se mogu pisati u T-SQL-u radi kreiranja i modifikovanja struktura baze, manipulacije nad podacima ali i implementiranja bezbedonosnih protokola i kreiranja rezervnih kopija podataka.

# Obrada upita

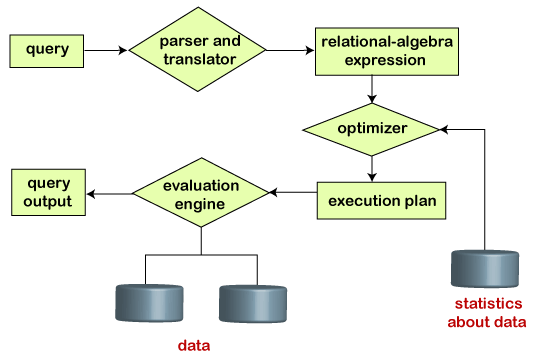
Obrada upita obuhvata kompajliranje i izvršavanje upita izraženih deklarativnim jezikom baza podataka kao što je strukturni jezik upita - SQL (eng. Structured Query Language). Obradu upita čine dve faze: faza kompajliranja i faza izvršenja. U fazi kompajliranja, kompajler prevodi upit u izvršni program i ovaj proces prevođenja, često nazvan i kompilacija upita, je sačinjen od leksičkih, sintaksnih i semantičkih analiza koje obrađuju specifikaciju upita ali i njegovu optimizaciju. Generisani kod obično čine fizički operatori za mašinu baze. Ovi operatori implementiraju način pristupa podacima, spojeve, projekcije, selekcije, grupisanja i agregaciju. U vremenu izvršenja, mehanizam baze podataka interpretira upite i izvršava program implementirajući dati upit kako bi proizveo rezultat.

U pitanju je aktivnost koja se izvršava prilikom uzimanja podataka iz baze i odvija se kroz nekoliko koraka:

1. Raščlanjivanje i prevođenje
2. Optimizacija
3. Evaluacija

Raščlanjivanje i prevođenje

Inicijalno, korisnički upit se izražava jezikom baze podataka višeg nivoa, kao što je SQL. Međutim, ljudima čitljiv SQL nije adekvatan za internu reprezentaciju upita sistemu. Za tako nešto se koristi relaciona algebra. Dakle, kada korisnik izvrši bilo kakav upit, radi generisanja interne forme datog upita, parser u sistemu provera sintaksu upita, proverava ime relacije u bazi, n-torku, i na kraju traženu vrednost. Parser kreira tzv. stablo upita i dalje to prevodi u oblik relacione algrebre. Zbog toga se obrada upita može prikazati kao na Slici 2.

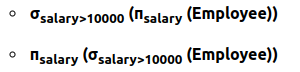


**Slika 2. Dijagram obrade upita**

Na primer, recimo da korisnik izvrši upit kojim želi da dobije sve zaposlene čija je plata veća ili jednaka 10000. Da bi ovo bilo moguće, izvršava se sledeći upit:

**select name from Employee where salary>10000;**

Da bi sistem razumeo šta se od njega traži, potrebno je prevesti upit u oblik relacione algrebre:



Nakon ovoga, svaka operacija algebre se izvršava korišćenjem različitih algoritama i na ovaj način otpočinje proces obrade upita.

Optimizacija

U ovoj fazi, procesor upita primenjuje pravila nad internim strukturama baze specificiranih upitom da bi ih pretvorio u ekvivalentne ali efikasnije reprezentacije. Ova pravila su zasnovana na matematičkim modelima relacione algebre i heuristici. Odabir adekvatnih pravila koja treba primeniti, kada i kako je funkcija mehanizma za optimizaciju.

Cena evaluacije upita varira za različite tipove upita. Pošto je sistem odgovoran za kreiranje evaluacionog plana, korisnik ne mora da vodi računa o ceni. Obično, sistem za upravljanje bazom generiše efikasan plan evaluacije upita čime minimizuje troškove. Ova vrsta zadatka koju izvršava sistem se naziva Optimizacija upita (eng. Query Optimization). Za optimizaciju, sistem mora imati procenjenu cenu analize svake operacije zbog toga što ukupna cena operacija zavisi od alokacije memorije za svaku od operacija, od cene njihovih pojedinačnih izvršenja itd.

Na primer, da bi se pronašli zaposleni sa platom većom ili jednakom 10000, može se koristiti index nad platom ali se može i analizirati tj. pretražiti kompletan dokument, što opet, zavisi od plana evaluacije.

Da bi se izabralo najoptimalnije rešenje, potrebno je izabrati onaj plan evaluacije koji ima najnižu cenu. Cena se procenjuje korišćenjem statističkih informacija iz kataloga baze podataka (broj n-torki u svakoj relaciji, veličina torki i sl.).

Cena zapravo predstavlja ukupno vreme potrebno da se odgovori na dati upit. U cenu ulazi dosta faktora koji uključuju i pristup disku, CPU vreme pa čak i mrežnu komunikaciju. Najčešće je presudan pristup disku i ujedno se i najlakše izračunava i obuhvata:

* Broj pretraživanja
* Broj pročitanih blokova
* Broj upisanih blokova

gde je cena upisa veća od cene čitanja bloka. Nakon upisa, podaci se čitaju da bi se obezbedilo da je upis prošao uspešno.

Evaluacija

Poslednji korak kod obrade upita jeste faza evaluacije. Najbolji kandidat plana evaluacije generisan od strane mehanizma za optimizaciju se u ovoj fazi bira i izvršava. Ovde se može primetiti postojanje nekoliko načina za izvršavanje upita. Osim izvršavanja na jednostavan sekvencijalni način, neke od metoda u upitu se mogu izvršiti i paralelno, kao nezavisni procesi ili kao međusobno zavisni procesi i niti. Nezavisno od izabranog načina, konačni rezultati moraju biti isti.

**Plan evaluacije:**

Najpre je potrebno kreirati plan evaluacije upita. Dodaci i beleške u planu se mogu odnositi na algoritme koji se koriste za neki određen indeks ili specifične operacije. Relaciona algebra sa takvim anotacijama se naziva Evaluacionim primitivama. Ove primitive nose instrukcije koje su potrebne za izvršavanje evaluacije, tako da plan definiše sekvencu primitivnih operacija potrebnu za evaluaciju upita. Plan se naziva još i plan izvršavanja upita. Mehanizam izvršavanja upita je odgovoran za generisanje rezultata tako što preuzme plan izvršenja, izvrši ga i kreira rezultat na osnovu toga.

# 

# MS SQL - Obrada upita

Mehanizam SQL baze podataka obrađuje Transact-SQL upite korišćenjem dva režima rada:

* Izvršavanje u režimu reda (eng. Row mode execution)
* Izvršavanje u režimu gomile (eng. Batch mode execution)

Row mode

Metod koji se koristi sa tradicionalnim relacionim DBMS-ovima, gde se podaci čuvaju u formatu reda. Kada se izvrši upit i pristupi se podacima iz tabele, operatori iz stabla izvršenja i njihova deca čitaju svaki zahtevan red, kroz sve kolone specifirane šemom. Iz svakog pročitanog reda, SQL Server zatim uzima kolone koje se traže u skupu rezultata, kao što je i zahtevano SELECT iskazom, JOIN predikatom ili filterima. Ovaj režim izvršenja je veoma efikasan u OLTP (eng. Online Transaction Processing), scenarijima ali je manje efikasan kada se prolazi kroz velike količine podataka, kao što je slučaj u scenarijima sa skladištenjem podataka (eng. Data Warehousing).

Batch mode

Režim izvršenja gomile je metod izvršavanja upita koji obrađuje nekoliko redova odjednom, otud i termin gomila. Svaka kolona iz gomile se čuva kao vektor u posebnom delu memorije, tako da je ovaj režim izvršenja zasnovan na vektorima. On takođe koristi i algoritme koji su optimizovani za procesor sa više jezgara i povećanu propusnost memorije koji se često nalaze na modernom hardveru. Ovaj režim radi nad kompresovanim podacima kada je to moguće i eliminiše operator paralelizma i zamene koji se koristi u režimu izvršenja reda. Rezultat jeste bolji paralelizam i brže performanse. Kada se upit izvršava i pristupa podacima iz kolona i njihovim indeksima, operatori iz stabla izvršenja i njihova deca čitaju više redova odjednom u segmentima kolona. SQL Server i u ovom režimu, čita samo one kolone koje su specificirane upitom.

Obrada SELECT iskaza u SQL-u

Obrada jednog iskaza je najosnovniji način na koji SQL izvršava upite, što je najbolje pokazati na primeru jednog SELECT iskaza.

Osnovni koraci koje SQL Server koristi za obradu SELECT iskaza uključuju sledeće:

1. Parser analizira iskaz i deli ga na logičke jedinice kao što su ključne reči, izrazi, operatori i identifikatori.
2. Kreira se stablo upita koje opisuje logičke korake koji si potrebni za transformaciju izvornih podataka u format koji je zahtevan kao rezultat.
3. Optimizator upita analizira različite načine za pristup tabelama. On zatim bira niz koraka koji najbrže vraćaju rezultat dok pritom koriste najmanje resursa. Stablo upita se ažurira kako bi sačuvalo tačne korake koji su izvršeni i poslednja optimizovana verzija stabla se naziva plan izvršenja (eng. execution plan).
4. Relacioni mehanizam zatim započinje plan izvršenja. Kako su koraci koji uzimaju podatke iz tabele u ovom trenutku već izvršeni, relacioni mehanizam (eng. Relational engine) traži od mehanizma skladištenja date podatke.
5. Relacioni mehanizam zatim obrađuje podatke koje je dobio i pretvara ih u traženi format i rezultat zatim vraća klijentu.

# MS SQL - Optimizacija upita

SQL Server koristi optimizator upita zasnovan na ceni tako što izračunava cenu nekoliko planova izvršenja, na osnovu statistike koju ima za kolone koje se traže i moguće indekse koji se mogu iskoristiti. Na osnovu ovih informacija, optimizator izračunava krajnju cenu svakog plana. Kompleksniji upiti mogu imati na hiljade mogućih planova izvršenja, međutim optimizator ne evaluira svaki mogući plan već koristi heuristiku radi odlučivanja koji od planova imaju najveće šanse za dobrim performansama. On će zatim, izabrati plan sa najnižom cenom evaluacije za dati upit.

Kada korisnik izvrši upit, odvija se sledeći proces:

1. Upit se prevodi u odgovarajuću sintaksu i generiše se stablo parsovanih objekata baze ukoliko je sintaksa upita korektna.
2. Dato stablo se koristi kao ulaz za mehanizam baze koji se naziva Algebrizer. U ovom koraku se vrši validacija kolona i objekata iz upita, potvrđuje se da oni postoje i identifikuju se njihovi tipovi podataka. Kao rezultat ovog koraka dobija se stablo obrade koje se koristi kao ulaz za sledeći korak.
3. Zbog toga što je optimizacija upita relativno skup proces gledano iz ugla procesora, mehanizam baze podataka kešira planove izvršenja u poseban deo memorije koji se naziva plan keša. Ukoliko plan za dati upit već postoji, biće preuzet iz keša. Upiti čiji su planovi već sačuvani u kešu imaju heš vrednost generisanu na osnovu T-SQL i naziva se heš upita (eng. query\_hash). Dakle, kada traži plan u kešu, mehanizam će generisati query\_hash a zatim pretražiti keš radi provere da li tražena vrednost već postoji.
4. U slučaju da plan ne postoji, optimizator upita koristi svoj mehanizam za generisanje nekoliko planova izvršenja zasnovan na statistici kolona, tabela i indeksa korišćenih u upitu, kao što je već i objašnjeno. Rezultat ovog koraka je plan izvršenja upita.
5. Upit se zatim izvršava korišćenjem datog plana i dobija se rezultat.

SELECT OrderDate, avg(SalesAmount)

FROM Sales

WHERE ShipDate = '2022-07-07'

GROUP BY orderdate;

GO

U datom primeru, SQL Server će proveriti postojanje OrderDate, ShipDate i SalesAmount kolona u tabeli Sales. Ako kolone postoje, on će dalje generisati heš vrednost za dati upit i pretražiti keš za vrednost koja se poklapa sa generisanom. Ukoliko je pronađe, pokušaće da iskoristi plan iz keša a u slučaju da takva vrednost ne postoji, pregledaće statistiku koju ima za OrderDate i ShipDate kolone. Klauzula WHERE koja se odnosi na ShipDate kolonu je predikat u ovom upitu i ukoliko postoji negrupisani indeks koji je uključuje, SQL Server će najverovatnije uključiti i taj indeks u plan ako je cena niža od one koja će koristiti grupisani indeks. Optimizator će zatim izabrati plan sa najnižom cenom i izvršiti ga.

Planovi upita uključuju kombinaciju relacionih operatora koji se koriste za dobijanje podataka ali i bitne informacije kao što je recimo procenjeni broj redova. Jos jedan element plana izvršenja je memorija koja je potrebna za tražene operacije. Memorija koja je potrebna upitu zove se dodeljena memorija (eng. memory grant). Dodeljena memorija je dobar primer važnosti statistike. Ukoliko SQL Server misli da će operator vratiti deset miliona redova kada on zapravo vraća samo stotinak, mnogo će veći deo memorije biti dodeljen upitu. Dodeljena memorija koja je veća od potrebne može izazvati dvostruki problem. Prvo, upit može naići na čekanje na semaforu (eng. RESOURCE\_SEMAPHORE wait), što znači da upit čeka da mu server dodeli veliku količinu memorije. Podrazumevano, SQL Server čeka 25 puta cena upita u sekundama pre izvršenja, do maksimuma od 24 časa. Drugo, kada se upit izvrši, ukoliko nema dovoljno dostupne memorije, upit će se prebaciti u tempdb što je mnogo sporije od rada u memoriji.

# Evaluacija operatora - Strategije

Upit se obično sastoji od nekoliko relacionih operatora. Odgovarajući plan fizičkog upita je sačinjen od više fizičkih operatora, a jedan fizički operator je zapravo implementacija relacionog operatora. Za svaki relacioni operator, postoji nekoliko alternativnih algoritama za njegovu implementaciju i evaluaciju, tako da je za performanse upita ključno izabrati odgovarajući fizički operator. Kako to učiniti zavisi od više faktora, kao što je postojanje indeksa, baferovanje i veličina dostupnog bafera, veličina ulaznih relacija i njihovo sortiranje itd.

Kada se koriste različiti operatori kao što su <, =, >, ≠ kao i njihove kombinacije, postoje razne tehnike koje DBMS koristi da bi ih evaluirao. Najčešće se koriste sledeće tri:

**Iteracije -** Ispitati sve n-torke tabele, jednu za drugom.

**Indeksiranje -** Ukoliko je u pitanju spoj (eng. join) ili selekcija i kolone koje se koriste imaju indeks, onda se taj indeks koristi za evaluaciju izraza radije nego iteracija svih zapisa.

**Particionisanje -** U pitanju je grupisanje skupa kolona u jedan blok nalik heširanju. Zapisi su obično sortirani i od toga se kreiraju grupe. Na primer, ukoliko postoje zapisi zaposlenih čije godine variraju od 22 do 55, oni se mogu sortirati na osnovu toga a zatim grupisati u 4 grupe: 22 - 30, 31 - 40, 41 - 50, 51 - 55. Tako da, ako se želi izvršiti pretraživanje bilo kog zapisa korišćenjem operatora, DBMS će prvo tražiti blok odgovarajuće particije a zatim iterirati kroz zapise u tom bloku. Dakle, korišćenjem sortiranja ili heširanja, ulazne n-torke se mogu particionisati i zameniti skuplji operator sličnim operacijama nad manjim ulazima.

Ako je tabela mala, metod iteracije je najefikasniji dok se za veće tabele indeksiranje ili particionisanje pokazalo efikasnijim.

Selekcija

U zavisnosti od postojanja indeksa i sortiranja, mogući su sledeći scenariji:

* Nema indeksa i podaci nisu sortirani: ukoliko data kolona ne sadrži indeks i njeni podaci nisu sortirani, jedino moguće rešenje jeste skeniranje i prolaz kroz ceo fajl, što je veoma sporo.
* Nema indeksa ali podaci jesu sortirani: radi se binarna pretraga fizičke organizacije fajla, tj. analiza sortirane datoteke. Brže od prethodnog scenarija ali ne može se računati da će relacija zauvek ostati sortirana.
* B+ stablo indeksa: za predikat[[2]](#footnote-2) selekcije nejednakosti, korišćenje B+ stabla indeksa je najbolja strategija. Za predikat selekcije jednakosti, heširanje je bolje rešenje. Treba primetiti da ukoliko indeksi nisu grupisani, cena vraćanja n-torki je viša, zbog toga što oni fizički nisu blizu što dalje iziskuje više cene ulaza i izlaza.
* Heš indeks: najbolja strategija za predikat selekcije jednakosti. Cena uključuje nekoliko I/O operacija za dobijanje stranica sa indeksa plus ulazno/izlazne operacije za vraćanje indeksa koji ispunjavaju uslov, što ponovo zavisi i od grupacije indeksa.

Za obradu generalne selekcije, predikati se iskazuju kao konjuktivna normalna forma, odnosno kolekcija konjukcija povezanih AND operatorom gde svaka sadrži uslove povezane OR operatorom. Za konjukcije koje sadrži OR operator se kaže da sadrže disjunkciju, na osnovu čega se mogu razmatrati dva sledeća scenarija:

* Nema disjunkcije: preuzeti podatke pretraživanjem datoteke (eng. file scan) ili pomoću jednog ili više indeksa.
* Ima disjunkcije: dok god ima disjunkcije, mora se vršiti pretraga datoteke dok god na jednom od uslova ne postoji indeks. U suprotnom, ako svaki uslov ima dostupan indeks, može se jednostavno uzeti unija kandidata n-torki.

## 

Projekcija

U relacionoj algebri, projekcija zahteva uklanjanje neželjenih atributa i eliminaciju duplih n-torki. Iako je relativno lako ukloniti neželjene atribute, eliminacija duplikata traži malo veći posao, ili sortiranje ili heširanje.

* Projekcija zasnovana na sortiranju: prvo se vrši sortiranje a zatim skeniranje tj. pretraga sortirane datoteke da bi se eliminisali duplikati. Vremenska kompleksnost je O(M \* logM), gde je M broj stranica. Usko grlo (eng. bottleneck) predstavlja korak sortiranja.
* Projekcija zasnovana na heširanju: vredi ukoliko postoji veliki broj baferovanih stranica **B**. Koristi se jedna stranica kao ulazni bafer i **B - 1** stranica kao izlazni bafer. Heš funkcija **f** particioniše n-torke izlaznim baferima. Drugim rečima, duplikati se mogu naći samo u okviru jedne particije. Tada, za svaku particiju nastalu u prvoj fazi, obrađuje se jedna stranica u trenutku. Za stranicu koja se trenutno obrađuje, treba pročitati sve uneto u memorijsku heš tablicu radi eliminisanja duplikata. Važno je primetiti da se treba koristiti heš funkcija drugačija od prethodne. Ova strategija neće funkcionisati kada je veličina heš tablice za particiju veća od broja dostupnih bafer stranica B (problem prelivanja particije - eng. partition overflow). Međutim, data particija se može podeliti na podparticije i zatim heširanje primeti rekurzivno.

Kada se ova dva pristupa uporede, pristup sa sortiranjem je efikasniji kada postoji veliki broj duplikata ili kada je raspoređivanje heš vrednosti neujednačeno. Korisna nuspojava je što će rezultat biti sortiran.

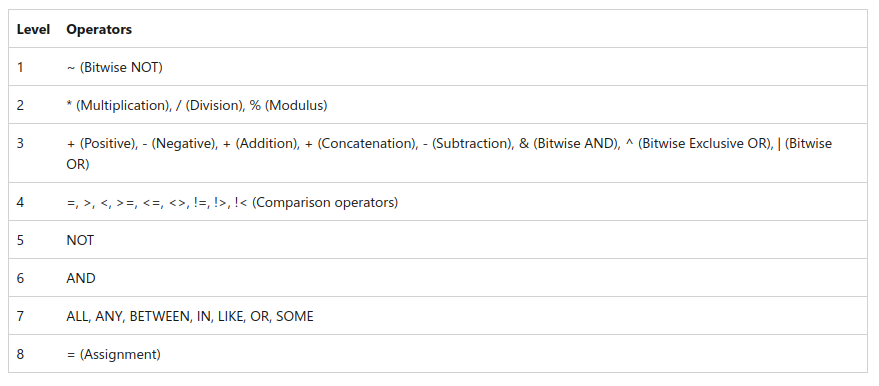
Prednosti baferovanja

U mnogo algoritama se evaluacija oslanja na baferovanje stranica. U slučaju da nekoliko operacija radi konkurentno, smanjuje se veličina dostupnog bafera. Isto tako, ukoliko se n-torkama pristupa pomoću negrupisanih indeksa, najčešće neće biti u baferu. Ovo je zbog toga što, u ovom kontekstu, svaka n-torka koja se povuče će najverovatnije dovesti i novu stranicu i bafer bi se vrlo brzo napunio. Kod recimo, indeksirane ugnježdene petlje spojeva, najbolje je sortirati spoljne relacije tako da ostane manje n-torki unutrašnje relacije (zbog toga što susedne spoljne n-torke verovatno pripadaju istoj particiji indeksa i stoga se tih nekoliko unutrašnjih relacija često pojavljuje). Samim tim je upravljanje baferom lakše na ovaj način.

# 

# Transact SQL - Evaluacija operatora

Kada kompleksan izraz ima više operatora, prvenstvo operatora odlučuje redosled operacija. Redosled izvršenja može značajno uticati na rezultujuću vrednost. Operatori imaju nivoe prvenstva prikazane u tabeli na Slici 3. Operator višeg nivoa biva evaluiran pre operatora nižeg nivoa. U datoj tabeli, nivo 1 je najviši nivo dok je nivo 8 najniži.



**Slika 3. Prvenstvo operatora T-SQL**

Kada dva operatora u iskazu imaju isti nivo prvenstva, evaluiraju se s leva na desno na osnovu njihove pozicije u izrazu. Na primer, u izrazu koji se koristi u sledećoj SET naredbi, operator oduzimanja se evaluira pre operatora sabiranja:

DECLARE @MyNumber INT;

SET @MyNumber = 4 - 2 + 27;

SELECT @MyNumber;

Ukoliko je potrebno pregaziti definisano prvenstvo, mogu se koristiti zagrade. Sve unutar zagrada se evaluira kao jedna vrednost koju može koristiti bilo koji operator van tih zagrada. Na primer, u sledećem SET iskazu, operator množenja ima prvenstvo nad operatorom sabiranja tako da se operacija množenja evaluira prva:

DECLARE @MyNumber INT;

SET @MyNumber = 2 \* 4 + 5;

SELECT @MyNumber;

Dodavanje zagrada utiče na to da se sabiranje evaluira prvo:

DECLARE @MyNumber INT;

SET @MyNumber = 2 \* (4 + 5);

SELECT @MyNumber;

U slučaju da iskaz ima ugnježdene zagrade, najdublja zagrada ima prednost i ona se evaluira prva. Na sledećem primeru iskaz **5 - 3** je unutar najdublje ugnježdene zagrade i on će biti evaluiran prvi, davajući rezultat **2**. Zatim se operator sabiranja evaluira dodavajući **4** rezultatu a na kraju se izračunata vrednost pomnoži sa **2**.

DECLARE @MyNumber INT;

SET @MyNumber = 2 \* (4 + (5 - 3) );

SELECT @MyNumber;

Dalje, ukoliko izraz sadrži kombinaciju logičkih operatora, redosled evaluacije je: **NOT, AND, OR.** Logički operatori u uslovnom iskazu nisu uvek evaluirani s leva na desno. Treba izbegavati korišćenje uslovnog izraza ako je njegova tačnost uslovljena redosledom kojim se logički izrazi evaluiraju. Na primer:

F2 / (NULLIF(F1,0)) > 500

F1 <> 0 AND F2/F1 > 500

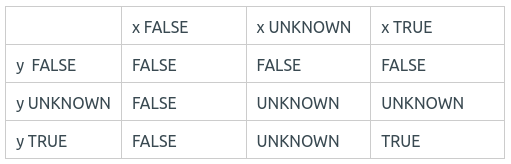
Prvi izraz garantuje isključenje deljenja nulom dok drugi dozvoljava mogućnost greške zbog toga što redosled njegove evaluacije određuje isključenje nula.

Rezultati logičke evaluacije

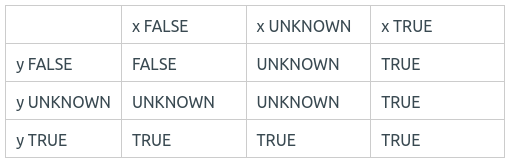
Svaki logički iskaz u uslovnom izrazu se evaluira jednim od 3 rezultata:

* TRUE
* FALSE
* UNKNOWN

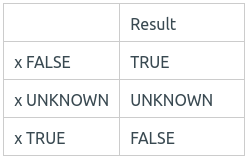
U sledećoj tabeli prikazana je AND logika koja se koristi kod evaluacije uslova pretrage:



U sledećoj tabeli prikazana je OR logika koja se koristi kod evaluacije uslova pretrage:



U sledećoj tabeli prikazana je NOT logika koja se koristi kod evaluacije uslova pretrage:



Ovo je najbolje objasniti na primeru jednog SELECT iskaza. Aritmetički i bitski operatori se evaluiraju pre logičkih operatora, kao što je već i objašnjeno. U sledećem primeru, uslov boje se odnosi na model 21 a ne 20 jer se AND operator evaluira pre i ima prednost nad OR operatorom:

SELECT ProductID, ProductModelID

FROM Production.Product

WHERE ProductModelID = 20 OR ProductModelID = 21

AND Color = 'Red';

GO

Da bi se forsirala prednost OR operatora, mogu se dodati zagrade. Sledeći upit pronalazi samo proizvode modela 20 i 21 koji su crvene boje:

SELECT ProductID, ProductModelID

FROM Production.Product

WHERE (ProductModelID = 20 OR ProductModelID = 21)

AND Color = 'Red';

GO

Korišćenje zagrada, čak i onda kada nisu obavezne, povećava čitljivost upita i smanjuje mogućnost greške zbog evaluacije operatora. Što se perfomansi tiče, nema neke značajne razlike sa ili bez korišćenja zagrada. Sledeći primer pokazuje bolje razumevanje originalnog upita iako su sintaksno isti:

SELECT ProductID, ProductModelID

FROM Production.Product

WHERE ProductModelID = 20 OR (ProductModelID = 21

AND Color = 'Red');

GO

Evaluacija iskaza u vremenu izvršenja

Tokom optimizacije SELECT iskaza u proceduri datoj u sledećem primeru, optimizator upita pokušava da evaluira očekivanu kardinalost rezultata za uslov OrderDate > @d+1. D je parametar čija je vrednost poznata u vreme optimizacije što dozvoljava optimizatoru da precizno proceni veličinu rezultata, što dalje pomaže pri izboru najboljeg plana upita:

USE AdventureWorks2014;

GO

CREATE PROCEDURE MyProc( @d datetime )

AS

SELECT COUNT(\*)

FROM Sales.SalesOrderHeader

WHERE OrderDate > @d+1;

Na sledećem primeru koji je sličan prethodnom, iskaz @d+1 je sada zamenila lokalna promenljiva @d2 i samim tim iskaz se evaluira kod SET naredbe umesto u samom upitu:

USE AdventureWorks2014;

GO

CREATE PROCEDURE MyProc2( @d datetime )

AS

BEGIN

DECLARE @d2 datetime

SET @d2 = @d+1

SELECT COUNT(\*)

FROM Sales.SalesOrderHeader

WHERE OrderDate > @d2

END;

Kada se SELECT naredba u MyProc2 optimizuje u SQL Serveru, vrednost @d2 nije poznata. Samim tim, optimizator upita koristi podrazumevanu procenu za selektivnost OrderDate > @d2, što je u ovom slučaju trideset procenata.

# 

# Literatura

* <https://www.sqlservertutorial.net/getting-started/what-is-sql-server/>
* <https://www.techtarget.com/searchdatamanagement/definition/SQL-Server>
* <https://yunpengn.github.io/blog/2019/01/05/relational-operators/#Selection>
* https://docs.microsoft.com/en-us/sql/t-sql/language-elements/operator-precedence-transact-sql?view=sql-server-ver15
* <https://docs.teradata.com/r/Teradata-VantageTM-SQL-Functions-Expressions-and-Predicates/March-2019/Logical-Predicates/Logical-Operators-and-Search-Conditions/NOT-Truth-Table>
* <https://docs.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide?view=sql-server-ver15#sql-statement-processing>
* <https://docs.microsoft.com/en-us/learn/modules/describe-sql-server-query-plans/2-describe-types>
* https://clas.iusb.edu/computer-science-informatics/research/reports/TR-20080105-1.pdf

1. eng. ACID - Atomicity, consistency, isolation and durability [↑](#footnote-ref-1)
2. Predikat - uslovni izraz koji evaluira boolean vrednost, može vratiti kao rezultat samo true/false vrednost. [↑](#footnote-ref-2)