DOI: 10.19279/TVZ.PD.2016-4-2-09

OPTIMIZACIJA UPITA U MICROSOFT SQL SERVER BAZI POMOĆU INDEKSA

MICROSOFT SQL SERVER QUERY OPTIMIZATIONS USING INDEXES

Mario Knok[1], Željko Kovačević[2]

[1]Tehničko veleučilište u Zagrebu

Sažetak

Sadržaj rada je objasniti strukturu indeksa Microsoft SQL Server baze kako bi se optimizirale tablice u bazi tako da se upiti nad njima izvršavaju u optimalnom vremenu. Ispravno postavljanje indeksa u tablicama jedno je od najbitnijih radnji vezanih za optimizaciju rada s bazom podataka, te se u ovom radu objašnjava na koji su način indeksi implementirani u SQL Serveru. Uvidom u tu strukturu, te serijom testova na kraju rada se donose opći zaključci o najboljoj praksi pri korištenju samih indeksa.

Abstract

The content of this paper is to explain the structure of the Microsoft SQL Server database indexes so that database tables could be optimized in order to maximize the performace of the queries executed on those tables.
Proper use of table indexes is one of the most important activities related to the database optimization, and this paper explains in detail of how the indexes are implemented in SQL Server. Using the knowlegde of this structure, along with a series of tests at the end of the paper will help to make general conclusions about best practices for the use of table indexes.

1. Uvod

1. *Introduction*

Microsoft SQL Server je relacijska[1] baza podatka razvijena od strane Microsofta. Prva verzija izašla je 1989. godine, a u trenutku pisanja posljednja izdana verzija je SQL Server 2014 (verzija 12.0), dok je najavljena verzija SQL Server 2016 (verzija 13.0). [1]

U svom dugogodišnjem iskustvu s izradom programa koji kao bazu podatka koriste Microsoft SQL Server često sam se susretao s problemom optimizacije tablica i upita nad tim tablicama. Cilj je da se upiti nad bazom izvršavaju u najkraćem vremenskom roku optimalno koristeći resurse servera. Optimizacije su najčešće bile povezane s pažljivom izradom indeksa u bazi. Zbog toga ću ovim radom detaljno objasniti sve načine kojima se tablice mogu optimizirati pomoću indeksa.
Za razumijevanje ove vrste optimizacije potrebno je razumjeti način na koji su indeksi implementirani u Microsoft SQL Serveru. Na početku rada prikazana je struktura indeksa u Microsoft SQL Serveru da bi saznali razloge zbog kojih se određene vrste optimizacije primijenjuju. Nakon toga napravit ćemo seriju testova kojima ćemo provjeravati parametre vezane za brzinu izvođenja upita ovisno o vrsti indeksa koju koriste.

2. Arhitektura indeksa u Microsoft SQL Serveru

2. *Microsoft SQL Server Index Architecture*

Microsoft SQL Server indekse implementira korištenjem strukture B-stabla[2].
Sami podaci se na disku spremaju u blokovima od 8 KB (8192 bajta) koji se nazivaju stranicama (eng. *pages*). Stranice su najmanje količine podataka koje SQL Server s diska učitava u memoriju, odnosno iz memorije sprema na disk. [2]
Stranice se sastoje od zaglavlja i tijela. Zaglavlje sadrži razne podatke o stranici, uključujući i vrstu

1 Više o tome na https://en.wikipedia.org/wiki/Relational_database_management_system.

2 Više o B-stablima na https://en.wikipedia.org/wiki/B-tree.

stranice, koja među ostalim može biti stranica s podacima (list B-stabla) ili indeksna stranica (unutrašnji čvorovi B-stabla). Tijelo stranice sadrži same zapise, te na kraju adrese tih zapisa. U pitanju je niz 2-bajtnih vrijednosti koje sadrže broj bajtova od početka stranice (eng. *offset*) na kojem počinje zapis. [2]
Zapisi (eng. *records*) u indeksnim stranicama sadrže ključ, te pokazivač na pod-stablo za taj ključ. Pokazivač može biti adresa indeksne stranice unutrašnjeg čvora ili adresa samog zapisa u podatkovnoj stranici, tj. listu stabla [3].
Zapisi u podatkovnim stranicama sadrže korisničke podatke iz tablica. Svaki zapis sadrži jedan red iz tablice. Jedan red podataka može zauzeti maksimalno 8060 bajtova. To ne uključuje LOB[3] tipove podataka. Oni se spremaju u za to posebne vrste stranica, LOB stranice. [4]
Također, ako zapis sadrži neki od varijablinih4 tipova podataka koji nisu LOB, te ako ukupna veličina reda prelazi 8060 bajtova, jedan ili više tih redova će također biti spremljeni u LOB stranice [4]. Tako da treba uzeti u obzir veličine stupaca u tablici budući da zahvaćanje podataka koji su spremljeni u više stranica zahtijeva čitanje dodatnih blokova s diska, te usporava upite.

2.1 Hrpa

2.1 *Heap*

Hrpa (eng. *heap*) je najjednostavnija struktura podataka u SQL Serveru; to je samo hrpa stranica povezana u jedan objekt. Hrpa je svaka tablica koja nema definiran klasterirani indeks. Podaci, tj. redovi u hrpi nisu spremljeni u nekom posebnom redosljedu. Ukoliko ima mjesta, SQL Server novi red sprema u neku postojeću stranicu, a ako nema, alocira novu stranicu i sprema ga u nju. [2]
Ako hrpu usporedimo s indeksima, ona je jednostavnija za održavanje, te brža pri ubacivanju redova jer ne mora održavati fizički redoslijed redova po nekom ključu [2]. Za primjer hrpe kreirat ćemo jednu tablicu u bazi i ubacit ćemo jedan red:

```
CREATE TABLE hrpa(naziv varchar(1000))
INSERT INTO hrpa VALUES('neki tekst')
```

Ukoliko pozovemo naredbu DBCC IND koja prikazuje popis stranica tablice:

```
DBCC IND(Diplomski, hrpa, -1)
```

vraća se ovaj rezultat (manje bitni stupci nisu prikazani):
Rezultat je vratio dvije stranice. Prva stranica ima *PageType 10*, te predstavlja IAM[5] stranicu. Jedina stranica s *PageType 1* (stranica s podacima) je stranica koja je u datoteci 1 pod brojem 321; označava se kao (1:321). Ukoliko izvršimo naredbu pregleda sadržaja te stranice:

```
DBCC PAGE(Diplomski, 1, 321, 1)
```

vraća se ovaj rezultat (manje bitni dijelovi su izrezani):

```
PAGE HEADER:
m_pageId = (1:321)
m_headerVersion = 1
m_type = 1

DATA:
Slot 0, Offset 0x60, Length 21, DumpStyle
BYTE
```

PageFID	PagePID	IAMFID	IAMPID	IndexID	PageType	IndexLevel	NextPageFID	NextPagePID	PrevPageFID	PrevPagePID
1	323	NULL	NULL	0	10	NULL	0	0	0	0
1	321	1	323	0	1	0	0	0	0	0

Tablica 1
Rezultat DBCC IND za hrpu

Table 1 *The output of DBCC IND command for the heap*

3 LOB (eng. large object) tipovi podataka su oni koji sadrže velike količine podataka, te se nikad ne spremaju u podatkovne stranice; u pitanju su tipovi text, ntext, image, nvarchar(max), varchar(max), varbinary(max) i xml.

4 To su tipovi čija veličina u različitim redovima iste tablice nije konstantna; u pitanju su tipovi varchar, nvarchar, varbinary i sql_variant.

5 IAM stranica (eng. index allocation map) sadrži popis stranica za određeni objekt u bazi.

```
Record Size = 21

00000000:   30000400 01000001 0015006e
656b6920 74656b73  0.........neki teks
00000014:   74                        t

OFFSET TABLE:
Row - Offset
0 (0x0) - 96 (0x60)
```

Na ispisu se vide razni podaci o stranici, te sadržaj zaglavlja i tijela stranice. U području DATA se vide podaci ubačenog reda u našoj hrpi. Vidi se da se nalazi na lokaciji 0x60 (heksadecimalni broj 60), tj. bajtu 96 i da je dugačak 21 bajt. U sirovom prikazu tog zapisa vidimo tekst koji smo upisali u njega. Na kraju se u području *OFFSET TABLE* nalazi popis adresa svih zapisa, a ovdje se ponovno vidi da je naš zapis na lokaciji 96, tj. na početku tijela stranice.
Za razliku od indeksa, hrpa nema ključ kojim se jednoznačno identificira neki red u tablici. Ako neklasterirani indeks ili strani ključ (eng. *foreign key*) treba spremiti adresu reda u hrpi, to radi tako da sprema 8-bajtni pokazivač na fizičku lokaciju reda u obliku (datoteka:stranica:mjesto). [2]

2.2 Klasterirani indeks

2.2 *Clustered Index*

Hrpe se koriste samo u vrlo specifičnim slučajevima. U praksi, skoro svaka tablica koja se koristi u SQL Serveru ima klasterirani indeks (eng. *clustered index*).
Klasterirani indeks koristi strukturu B-stabla gdje su redovi tablice spremljeni u listove stabla [2]. Kreirat ćemo tablicu iz koda ispod te ćemo vidjeti kako izgledaju stranice te tablice.

```
CREATE TABLE tablica(id int NOT NULL,
naziv char(2000))
CREATE UNIQUE CLUSTERED INDEX IX_tablica
ON tablica(id)
INSERT INTO tablica VALUES(10, 'a')
INSERT INTO tablica VALUES(5, 'b')
INSERT INTO tablica VALUES(20, 'c')
INSERT INTO tablica VALUES(30, 'd')
INSERT INTO tablica VALUES(40, 'e')
```

Prva stranica je IAM stranica, druga i četvrta su stranice s podacima, dok je treća (1:342) indeksna stranica (*PageType=2*). Indeksna stranica je korijen B-stabla a to se vidi jer jedina ima *IndexLevel 1*. Možemo uočiti da listovi pokazuju jedan na drugoga (stupci *NextPageFID* i *NextPagePID*). Prvi list sadrži redove s vrijednostima "a", 'b", "c" i "d", dok drugi sadrži red "e". Indeksna stranica sadrži dva zapisa vrste *INDEX_RECORD*, svaki veličine 11 bajtova:
Na taj način možemo vidjeti sadržaj korijena B-stabla našeg klasteriranog indeksa. Vidimo da prvi zapis (Row=0) ima ključ (*id (key)*) *NULL* i pokazuje na stranicu (1:321), dok drugi ima ključ 40 i pokazuje na (1:3528). Ako u tablicu ubacimo ovaj novi red, prikazuje nam se novi popis stranica prikazan u tablici 4.

Tablica 2 *Rezultat DBCC IND za klasterirani indeks*

Table 2 *The output of DBCC IND command for the clustered index*

PageFID	PagePID	IAM-FID	IAM-PID	In-dexID	PageType	IndexLe-vel	NextPage-FID	NextPa-gePID	PrevPa-geFID	PrevPa-gePID
1	323	NULL	NULL	1	10	NULL	0	0	0	0
1	321	1	323	1	1	0	1	3528	0	0
1	342	1	323	1	2	1	0	0	0	0
1	3528	1	323	1	1	0	0	0	1	321

Tablica 3 *Rezultat DBCC PAGE za indeksnu stranicu*

Table 3 *The output of DBCC PAGE command for the index page*

FileId	PageId	Row	Level	ChildFileId	ChildPa-geId	id (key)	KeyHashVa-lue	Row Size
1	342	0	1	1	321	NULL	NULL	11
1	342	1	1	1	3528	40	NULL	11

Tablica 4 *Popis stranica nakon ubacivanja reda u klasteriranu tablicu*

Table 4 *The list of pages after inserting a row into the clustered table*

PageFID	PagePID	IAMFID	IAMPID	IndexID	PageType	IndexLevel	NextPageFID	NextPagePID	PrevPageFID	PrevPagePID
1	323	NULL	NULL	1	10	NULL	0	0	0	0
1	321	1	323	1	1	0	1	3529	0	0
1	342	1	323	1	2	1	0	0	0	0
1	3528	1	323	1	1	0	0	0	1	3529
1	3529	1	323	1	1	0	1	3528	1	321

Tablica 5 *Sadržaj korijena nakon ubacivanja reda u klasteriranu tablicu*

Table 5 *The contents of the root node after inserting a row into the clustered table*

FileId	PageId	Row	Level	ChildFileId	ChildPageId	id (key)	KeyHashValue	Row Size
1	342	0	1	1	321	NULL	NULL	11
1	342	1	1	1	3529	20	NULL	11
1	342	2	1	1	3528	40	NULL	11

```
INSERT INTO tablica VALUES(25, 'f')
```

Ako pogledamo sadržaje listova, vidimo da su u stranici (1:321) ostali samo redovi s ključevima 5 i 10, nova stranica (1:3529) sadrži ključeve 20, 25 i 30, a stranica (1:3528) je ostala nepromijenjena. U sadržaju korijena vidimo da je ubačen srednji ključ: Ovdje možemo uočiti dva problema. Prvo, fizički redosljed stranica ne prati njihov logički redosljed. Vidimo da listovi idu 321 → 3529 → 3528. To se zove vanjska fragmentacija indeksa i povećava vrijeme pristupa disku, jer se pri skeniranju tablice po disku čita naprijed-natrag [5].
Drugi problem je taj da u stanicama ostaje prazan prostor. Stranica (1:321) sadrži samo dva a (1:3529) samo tri reda, a znamo da u stranicu stanu četiri ovakva reda. To se zove unutrašnja fragmentacija indeksa i također povećava vrijeme pristupa disku jer indeks zauzima više stranica nego je to potrebno [5].
Do fragmentacije je došlo jer smo u indeks ubacivali ključeve koji nisu išli po redu.
U klasteriranom indeksu je bitno da se ključevi ubacuju po veličini, jer na taj način izbjegavamo fragmentaciju indeksa [5]. Indeks se defragmentira ovom naredbom:

```
ALTER INDEX IX_tablica ON tablica REBUILD
```

Nakon toga ponovno provjeravamo popis stranica i vidimo da se broj stranica smanjio, te da su listovi po redu.
Ključevi klasteriranog indeksa bi trebali biti statični, tj. ne bi se naknadno smjeli mijenjati, jer njihova promjena također dovodi do fragmentacije.
Budući da se ključ klasteriranog indeksa kopira u svaki neklasterirani indeks, poželjno je da ključ klasteriranog indeksa zauzima što manje mjesta kako bi i neklasterirani indeksi bili na što manje stranica.
Također, ukoliko kreiramo tablicu s klasteriranim indeksom koji sadrži duple ključeve, oni će zauzimati više mjesta na disku jer se svakom nejedinstvenom ključu dodaje 4-bajtni podatak koji se zove *uniquifier* a služi da bi napravio razliku između dva jednaka ključa [5]. Kako je cilj da ključ klasteriranog indeksa bude što manji, najbolje bi bilo da se za klasterirane indekse koriste jedinstveni ključevi.

2.3 Neklasterirani indeks

2.3 *Nonclustered Index*

Tablica može imati samo jedan klasterirani indeks. Za ubrzavanje učitavanja podataka pomoću drugih stupaca u tablici koriste se neklasterirani indeksi (*eng. non-clustered index*). Tablica može imati više neklasteriranih indeksa odjednom.
Strukturno gledajući, neklasterirani indeksi se ne razlikuju od klasteriranih indeksa; i jedni i drugi se spremaju u obliku B-stabla. Razlika je u tome da su listovi neklasteriranog indeksa također indeksne stranice, dok su

kod klasteriranog indeksa listovi stranice s podacima. [6]
Svaki put dok se podaci traže preko neklasteriranog indeksa, ostali stupci se traže preko klasteriranog indeksa. Ukoliko se za upite na tablici često traže samo određeni stupci, oni se mogu uključiti u sadržaj neklasteriranog indeksa, te dodatno traženje preko klasteriranog indeksa neće biti potrebno [2]. Takvi indeksi se zovu indeksi s uključenim stupcima (*eng. included columns*).

3. Rezultati mjerenja izvršavanja SQL upita

3. *The Results of the SQL Query Performance Tests*

3.1 Usporedba hrpe i klasterirane tablice

3.1 *Comparison between the Heap and the Clustered Table*

Kreirat ćemo dvije tablice s jedinom razlikom da jedna ima klasterirani indeks po stupcu id, a druga nema:

```
CREATE TABLE hrpa(id int NOT NULL, broj
int, tekst char(1000))
```

```
CREATE TABLE tablica(id int NOT NULL,
broj int, tekst char(1000))
CREATE UNIQUE CLUSTERED INDEX IX_tablica
ON tablica(id)

INSERT INTO tablica(id, broj, tekst)
SELECT n, n, CAST(n AS varchar(10))
FROM (SELECT TOP 1000000 ROW_NUMBER()
OVER(ORDER BY v.number) AS n
      FROM master..spt_values v CROSS
      JOIN master..spt_values) numbers
```

U tablici 6 vidimo razlike između te dvije tablice. Kod ubacivanja podataka može se vidjeti prednost ubacivanja u hrpu, gdje je milijun redova ubačeno za upola manje vremena od klasterirane tablice, jer hrpa ne treba održavati B-stablo. Klasterirana tablica ima više stranica, a razlika je samo u indeksnim stranicama klasteriranog indeksa; podatkovne stranice su jednake u obje tablice.
Po planovima izvršavanja se može se vidjeti da se kod hrpe uvijek koristi skeniranje (table scan), dok se kod klasterirane tablice po stupcu id koristi traženje (clustered index seek). To se vidi i po vremenu koje se potrošilo kod izvršavanja upita koje je u ovom slučaju strogo vezano za broj pročitanih stranica.
Kod hrpe se uvijek mora čitati cijela tablica, dok

	Hrpa	Klasterirana tablica
Ubacivanje 1.000.000 redova	8.205 ms	16.224 ms
Broj stranica za 1.000.000 redova	142.859	143.390
Traženje reda po stupcu id (SELECT * FROM tablica WHERE id = 500)		
Plan izvršavanja		
Tablica od 1.000 redova	143 reads; 0 ms	2 reads; 0 ms
Tablica od 10.000 redova	1.429 reads; 15 ms	3 reads; 0 ms
Tablica od 100.000 redova	14.286 reads; 31 ms	3 reads; 0 ms
Tablica od 1.000.000 redova	142.858 reads; 218 ms	3 reads; 0 ms
Traženje reda po stupcu broj (SELECT * FROM tablica WHERE broj = 500)		
Plan izvršavanja	Identično kao po stupcu id	
Tablica od 1.000 redova		145 reads; 0 ms
Tablica od 10.000 redova		1.435 reads; 15 ms
Tablica od 100.000 redova		15.737 reads; 32 ms
Tablica od 1.000.000 redova		145.508 reads; 266 ms

Tablica 6
Usporedba hrpe i klasterirane tablice

Table 6
Comparison between the heap and the clustered table

se kod klasterirane tablice traži po B-stablu, te se čitaju samo potrebne indeksne stranice, a na kraju i sama stranica s podacima. Kod klasterirane tablice s 1.000 redova čitane su samo dvije stranice, jer ta tablica ima samo korijen i listove, a tablice s više redova sadrže i unutrašnje čvorove stabla, te su imale po tri čitanja.
Kreirat ćemo hrpu i klasteriranu tablicu s kojima ćemo mjeriti brzinu ubacivanja podataka s različitim brojem neklasteriranih indeksa:

```
CREATE TABLE hrpa(id INT NOT NULL, a INT,
b INT, c INT, d INT, e INT)
CREATE TABLE tablica(id INT NOT NULL, a
INT, b INT, c INT, d INT, e INT)
CREATE UNIQUE CLUSTERED INDEX IX_tablica
ON tablica(id)

INSERT INTO tablica(id, a, b, c, d, e)
SELECT n, n, n, n, n, n
FROM (SELECT TOP 100000 ROW_NUMBER()
OVER(ORDER BY v.number) AS n
       FROM master..spt_values v CROSS
       JOIN master..spt_values) numbers
```

Mjerit ćemo ubacivanje na hrpi i klasteriranoj tablici, te ćemo mijenjati broj neklasteriranih indeksa u svakoj od njih i ponoviti mjerenje.

Tablica 7 *Rezultati mjerenja ubacivanja u tablice s različitim brojem indeksa*

Table 7 *Insertion performance of tables with increasing number of indexes*

	Hrpa	Klasterirana tablica
Ubacivanje bez neklasteriranih indeksa	328 ms	593 ms
Ubacivanje s 1 neklasteriranim indeksom	890 ms	1.106 ms
Ubacivanje s 2 neklasterirana indeksa	1.810 ms	1.996 ms
Ubacivanje s 3 neklasterirana indeksa	2.340 ms	2.481 ms
Ubacivanje s 4 neklasterirana indeksa	2.917 ms	3.058 ms
Ubacivanje s 5 neklasteriranih indeksa	3.338 ms	3.681 ms

Po izmjerenim podacima se vidi da je ubacivanje u hrpu nešto brže od ubacivanja u klasteriranu tablicu budući da klasterirana tablica uvijek ima jedan indeks više od hrpe. Također se vidi da se trajanje ubacivanja linearno povećava s povećanjem broja neklasteriranih indeksa u tablici. Za svaki dodatni indeks potrebno je potrošiti dodatno vrijeme za održavanje B-stabla.

3.2 Jedinstvenost klasteriranog indeksa

3.2 *Uniqueness of Clustered Index Keys*

Napravit ćemo tablice s kojima ćemo provjeriti koliko se povećava veličina indeksa s povećanjem broja duplih ključeva u klasteriranom indeksu. Gledat ćemo broj stranica klasteriranog i neklasteriranog indeksa u istoj tablici, te usporediti s tablicom u kojoj je klasterirani indeks jedinstven.

```
CREATE TABLE jed(id int NOT NULL, broj
int NOT NULL)
CREATE UNIQUE CLUSTERED INDEX IX_jed ON
jed(id)
CREATE NONCLUSTERED INDEX IX_jed_broj ON
jed(broj)

CREATE TABLE nejed(id int NOT NULL, broj
int NOT NULL)
CREATE CLUSTERED INDEX IX_nejed ON
nejed(id)
CREATE NONCLUSTERED INDEX IX_nejed_broj
ON nejed(broj)
```

Obje tablice ćemo prvo popuniti s jedinstvenim vrijednostima, a nakon toga ćemo popunjavati tablicu s nejedinstvenim vrijednostima na ovaj način:

```
INSERT INTO nejed(id, broj)
SELECT FLOOR((n-1)*0.5), n
FROM (SELECT TOP 1000000 ROW_NUMBER()
OVER(ORDER BY v.number) AS n
       FROM master..spt_values v CROSS
       JOIN master..spt_values) numbers
```

U kodu je primjer faktora 0,5 koji će popuniti tablicu tako da je svaka druga vrijednost ponovljena. Rezultati mjerenja se nalaze u tablici 8.

Tablica 8 *Rezultat ponavljanja ključa klasteriranog indeksa*

Table 8 *The result of using duplicate keys for the clustered index*

	Broj stranica klasteriranog indeksa	Broj stranica neklasteriranog indeksa
Tablica s jedinstvenim klasteriranim indeksom	2.110	1.740
Tablica s nejedinstvenim klasteriranim indeksom s jedinstvenim ključevima	2.110	1.740
Jedan dupli ključ u njih deset (faktor 0.9)	2.211	1.841
2 dupla u 10 (faktor 0.8)	2.309	1.942
3 dupla u 10 (faktor 0.7)	2.415	2.041
4 dupla u 10 (faktor 0.6)	2.510	2.144
Svaki drugi je dupli (faktor 0.5)	2.612	2.241
Tri jednaka ključa za redom (dijelimo s 3)	2.778	2.409
Četiri jednaka ključa za redom (dijelimo s 4)	2.853	2.494
Pet jednakih ključeva za redom (dijelimo s 5)	2.917	2.542
Deset jednakih ključeva za redom (dijelimo s 10)	3.012	2.643
Sto jednakih ključeva za redom (dijelimo sa 100)	3.104	2.730
Tisuću jednakih ključeva za redom (dijelimo s 1.000)	3.112	2.738
Svi ključevi jednaki (faktor 0)	3.115	2.738

Prvo što se može uočiti je ukoliko su svi ključevi u tablici jedinstveni, vrsta klasteriranog indeksa ne utječe na broj stranica. Ukoliko je indeks nejedinstven, vidimo da kako se povećava broj ponavljajućih ključeva, tako raste broj stranica koje su potrebne za spremanje indeksa.

3.3 Usporedba različitih tipova podataka u ključevima

3.3 *Comparison of the Keys with Different Data Types*

Da bi uspoređivali performanse različitih tipova korištenih u ključevima indeksa, napravit ćemo ovakve tablice:

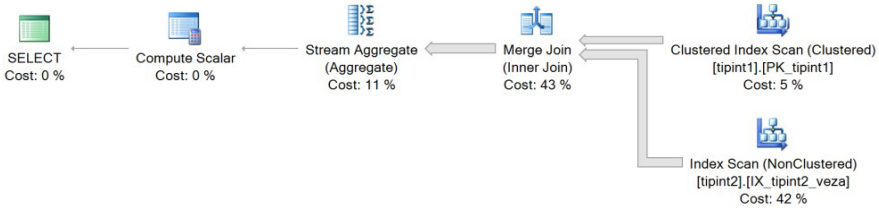
```
CREATE TABLE tipint1(id1 INT CONSTRAINT
PK_tipint1 PRIMARY KEY, broj INT)
CREATE TABLE tipint2(id2 INT CONSTRAINT PK_
tipint2 PRIMARY KEY IDENTITY, id1_veza INT)
CREATE NONCLUSTERED INDEX IX_tipint2_veza
ON tipint2(id1_veza)

INSERT INTO tipint1(id1, broj)
SELECT n, n%100
FROM (SELECT TOP 100000 ROW_NUMBER()
OVER(ORDER BY v.number) AS n
FROM master..spt_values v CROSS JOIN
master..spt_values) numbers

INSERT INTO tipint2(id1_veza)
SELECT (n+9)/10
FROM (SELECT TOP 1000000 ROW_NUMBER()
OVER(ORDER BY v.number) AS n
FROM master..spt_values v CROSS JOIN
master..spt_values) numbers
```

Tablica 9 *Rezultati mjerenja različitih tipova podataka u ključevima*

Table 9 *Measurement results for different data types inside keys*

Tip ključa i veze	int	real	bigint	float	money	datetime	unique-identifier	numeric (8)	numeric (38)	char(8)	char(8) bin
Veličina ključa (bajtova)	4	4	8	8	8	8	16	5	17	8	8
Plan izvršavanja											
Broj čitanih stranica	1.955	1.955	2.506	2.506	2.506	2.506	3.596	2.093	3.751	2.506	2.501
Vrijeme izvođenja (ms)	328	328	328	328	328	328	343	359	359	764	343

```
SELECT Sum(broj) FROM tipint1 JOIN
tipint2 ON id1=id1_veza
```

Ovo je primjer tablica gdje za ključeve koristimo tip int. Prva tablica će biti popunjena ključevima od 1 do 100.000. Druga će imati 10 puta više redova gdje će svaki ključ iz prve tablice biti spomenut 10 puta u stupcu id1_veza, te će na tom stupcu biti neklasterirani indeks. Mjerit ćemo upit prikazan na kraju koda.
Brojčani tipovi su svi jednako brzi osim tipa numeric koji je samo malo sporiji. Na performanse brojčanih tipova uopće ne utječe veličina ključa. Bitnije usporavanje se vidi tek na tipu char u slučaju kad koristi kodnu stranicu. U slučaju binarnog kodiranja6, char je također brz kao i numerički tipovi.

3.4 Usporedba tipova char i varchar u ključevima

3.4 *Comparison between Char and Varchar Data Types in the Index Keys*

Za ovu usporedbu smo kreirali jednake tablice s ključevima tipa char i varchar kao i u prošlom poglavlju, te smo samo mijenjali veličine tih ključeva. Primjer za char duljine 100 znakova (funkcija dbo.slova pretvara broj u abecedni tekstualni zapis):

```
CREATE TABLE char1001(id1 char(100)
CONSTRAINT PK_char1001 PRIMARY KEY,broj
INT)
CREATE TABLE char1002(id2 INT CONSTRAINT
PK_char1002 PRIMARY KEY IDENTITY, id1_
veza char(100))
CREATE NONCLUSTERED INDEX IX_char1002_
veza ON char1002(id1_veza)

INSERT INTO char1001(id1, broj)
SELECT REPLICATE('A', 100-4) + dbo.
slova(n-1), n%100
FROM (SELECT TOP 100000 ROW_NUMBER()
OVER(ORDER BY v.number) AS n
FROM master..spt_values v CROSS JOIN
master..spt_values) numbers

INSERT INTO char1002(id1_veza)
SELECT REPLICATE('A', 100-4) + dbo.
slova((n-1)/10)
FROM (SELECT TOP 1000000 ROW_NUMBER()
OVER(ORDER BY v.number) AS n
FROM master..spt_values v CROSS JOIN
master..spt_values) numbers

SELECT Sum(broj) FROM char1001 JOIN
char1002 ON id1=id1_veza
```

6 Binarno kodiranje ne koristi kodnu stranicu (npr. Croatian_CI_AS) za usporedbu i sortiranje tekstova.

Tablica 10 *Rezultati mjerenja s ključevima tipa char i varchar različitih veličina*

Table 10 *The performance of using char and varchar data types in the index keys*

	char	varchar	char bin	varchar bin
Veličina ključa 4 bajta	601 ms	624 ms	343 ms	359 ms
Veličina ključa 8 bajtova	764 ms	780 ms	343 ms	359 ms
Veličina ključa 16 bajtova	1.076 ms	1.092 ms	358 ms	374 ms
Valičina ključa 40 bajtova	2.028 ms	2.082 ms	382 ms	395 ms
Valičina ključa 100 bajtova	4.400 ms	4.431 ms	437 ms	445 ms

Vidimo da se povećanjem veličine ključa linearno povećava vrijeme izvršavanja upita. Ukoliko nam nije bitno uspoređivanje ili sortiranje teksta korištenjem kodne stranice, najbolje je staviti binarno kodiranje jer se time drastično povećavaju performanse indeksa, pogotovo kako se povećava veličina ključa.

3.5 Usporedba jednostavnih i složenih indeksa

3.5 *Comparison between Simple and Composite Indexes*

Jednostavni indeks je indeks čiji ključ sadrži samo jedan stupac, dok je složeni indeks onaj čiji se ključ sastoji od više stupaca. Kod korištenja složenih indeksa nedostatak je taj da se ključ indeksa povećava, te se time upiti generalno usporavaju. Napravit ćemo test gdje ćemo usporediti brzinu izvođenja upita u različitim varijacijama jednostavnih i složenih indeksa. Za izradu tablica sa složenim indeksima koristit ćemo ovakav kod:

```
CREATE TABLE tip2int1(id1 INT NOT NULL,
id2 INT NOT NULL, broj INT,CONSTRAINT PK_
tip2int1 PRIMARY KEY CLUSTERED(id1, id2))
CREATE TABLE tip2int2(id INT CONSTRAINT
PK_tip2int2 PRIMARY KEY IDENTITY, id1_
veza INT, id2_veza INT)
CREATE NONCLUSTERED INDEX IX_tip2int2_
veza ON tip2int2(id1_veza, id2_veza)

INSERT INTO tip2int1(id1, id2, broj)
SELECT n/1000, n%1000, n%100
FROM (SELECT TOP 100000 ROW_NUMBER()
OVER(ORDER BY v.number) AS n
FROM master..spt_values v CROSS JOIN
master..spt_values) numbers

INSERT INTO tip2int2(id1_veza, id2_veza)
SELECT (n+9)/10/1000, (n+9)/10%1000
FROM (SELECT TOP 1000000 ROW_NUMBER()
OVER(ORDER BY v.number) AS n
FROM master..spt_values v CROSS JOIN
master..spt_values) numbers

SELECT Sum(broj) FROM tip2int1 JOIN
tip2int2 ON id1=id1_veza AND id2=id2_veza
```

Tablica 11 *Rezultati mjerenja složenih indeksa*

Table 11 *The performance of composite indexes*

Broj stupaca i tip ključa	Broj čitanih stranica	Vrijeme izvođenja
2 smallint	1.955	593 ms
2 int	2.506	593 ms
2 bigint	3.607	593 ms
2 char(4)	2.506	1.334 ms
2 char(4) bin	2.500	780 ms
3 smallint	2.227	843 ms
3 int	3.052	843 ms
3 bigint	4.893	843 ms
3 char(4)	3.052	1.919 ms
3 char(4) bin	3.042	1.185 ms

Prvo što se iz rezultata može zaključiti je da za numeričke tipove, kao i kod jednostavnih indeksa, brzina izvođenja upita ne ovisi o veličini ključa. Najbitnije kod ovog testa je što vidimo da korištenje složenih indeksa usporava upit bez obzira na veličinu ključa. Ukoliko nam je najbitnija brzina izvođenja upita, te ako pri dohvaćanju uvijek koristimo cijeli ključ, preporuča se spojiti dva manja ključa u jedan veći. Npr. upit s dva stupca tipa int se izvršava za 593 ms, dok se upit s jednim stupcem bigint izvršava za 328 ms (izmjereno u poglavlju 3.3). Oba ključa su velika 8 bajtova, te u njih stanu jednaki podaci.

Ako nemamo upite koji dohvaćaju podatke koristeći samo jedan od dva stupca int-a, tada je bolje za ključ koristiti jedan stupac tipa bigint.

3.6 Utjecaj fragmentacije na brzinu skeniranja tablice

3.6 *The Impact of Fragmentation on the Table Scan Performance*

Za ovu provjeru napravit ćemo tablicu koja će prvo biti fragmentirana, te ćemo je nakon mjerenja defragmentirati, pa ponovno mjeriti. Fragmentiranu tablicu ćemo napraviti tako da će njen klasterirani indeks biti popunjen s milijun brojeva u obrnutom redoslijedu od normalnog smjera rasta ključeva, te ćemo još promijeniti slučajne ključeve u indeksu:

```
CREATE TABLE tablica(id int CONSTRAINT
PK_tablica PRIMARY KEY, broj int, tekst
char(1000))

DECLARE @a as INT = 1000000
WHILE @a > 0
BEGIN
      INSERT INTO tablica(id, broj,
      tekst) VALUES(@a, @a, 'A')

      SET @a = @a - 1
END
GO

UPDATE tablica SET id=-id WHERE
id%1000=FLOOR(RAND()*1000)GO 50
```

```
SELECT Count(broj) FROM tablica
```

Upit ćemo pokretati dvaput: prvi put dok stranice još nisu prethodno učitane u memoriju, te drugi put dok već jesu u memoriji. Sve to ćemo ponoviti više puta zbog veće preciznosti rezultata. Prethodno učitane stranice se brišu iz memorije na ovaj način:

```
CHECKPOINT
DBCC DROPCLEANBUFFERS
```

Ukoliko usporedimo fragmentiranu i nefragmentiranu tablicu, vidimo da postoje prilično velike razlike u rezultatima. Broj stranica kod fragmentirane tablice je veći jer sadrži više polupraznih stranica. Fragmentacija indeksa dakle utječe na performanse skeniranja tablice.

3.7 Performanse indeksa s uključenim stupcima

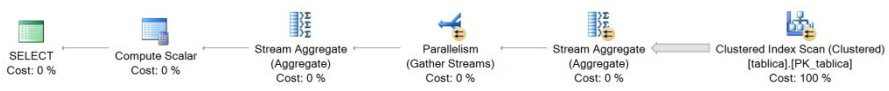
3.7 *The performance of Indexes with Included Columns*

Neklasterirani indeksi s uključenim stupcima (eng. included columns) sadrže i dodatne stupce koji ne pripadaju tom ključu. Uključeni stupci služe, ukoliko se tablica pretražuje putem tog indeksa, da bi se do tih stupaca došlo bez dodatnog traženja po klasteriranom indeksu. Izrada tablica koje ćemo testirati:

```
CREATE TABLE tablica(id INT CONSTRAINT
PK_tablica PRIMARY KEY,vrsta INT, broj
```

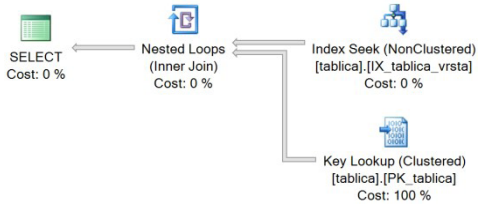
Tablica 12 *Rezultati mjerenja fragmentirane i nefragmentirane tablice*

Table 12 *The performance results of the fragmented and consolidated table*

	Logical reads	Physical reads	Read-ahead reads	CPU time	Elapsed time
Plan izvršavanja					
Prvo izvršavanje fragmentirane tablice	155.405	1	155.278	670 ms	4.506 ms
Drugo izvršavanje fragmentirane tablice	155.421	1.596	66.120	483 ms	1.806 ms
Prvo izvršavanje nefragmentirane tablice	143.662	1	143.261	515 ms	4.145 ms
Drugo izvršavanje nefragmentirane tablice	144.878	0	0	406 ms	329 ms

Tablica 13 *Rezultati mjerenja tablice s indeksom s uključenim stupcem*

Table 13 *The performance results of index with included columns*

	Bez uključenog stupca	S uključenim stupcem broj
Broj stranica	4.351	4.850
SELECT vrsta, broj FROM tablica WHERE vrsta<200		
Plan izvršavanja		
Broj čitanih stranica	6.110	8
Vrijeme izvršavanja	16 ms	0 ms
SELECT vrsta, broj FROM tablica WHERE vrsta<10000		
Plan izvršavanja		
Broj čitanih stranica	2.609	227
Vrijeme izvršavanja	62 ms	32 ms

```
INT)
CREATE NONCLUSTERED INDEX IX_tablica_
vrsta ON tablica(vrsta)INCLUDE(broj)
INSERT INTO tablica(id, vrsta, broj)
SELECT n, (n+9)/10, n%100
FROM (SELECT TOP 1000000 ROW_NUMBER()
OVER(ORDER BY v.number) AS n
FROM master..spt_values v CROSS JOIN
master..spt_values) numbers
```

Ukoliko usporedimo dvije tablice, vidi se da se performanse upita bitno povećavaju s uvođenjem uključenog stupca u indeks. Koristi se samo jedno brzo pretraživanje indeksa s čime se broj čitanih stranica drastično smanjuje, a tako i vrijeme izvršavanja upita. Ukoliko se često izvršava neki upit kojem se pristupa putem neklasteriranog indeksa, a traže se podaci i iz drugih stupaca, tada se oni preporučuju uključiti u taj indeks.

4. Zaključak

4. *Conclusion*

U radu je objašnjena arhitektura indeksa u SQL Serveru, te uz pojmove kao što su stranica i zapis, opisano je na koji način se podaci iz korisničkih tablica organiziraju i spremaju na disk. Detaljno su opisani pojmovi hrpe, klasteriranog i neklasteriranog indeksa. Zatim je napravljena serija testova iz kojih se mogu donijeti ovi zaključci vezani za optimizaciju upita u Microsoft SQL Serveru:

1. Stupci u tablicama ne trebaju biti veći od potrebnog jer bi zapisi na taj način zauzimali više stranica, te bi se nepotrebno povećavala količina čitanja s diska.
2. Hrpa se koristi samo u slučaju kad je najbitnije brzo ubacivanje podataka u tablicu, te dok brzina čitanja tih podataka nije toliko bitna.
3. Klasterirani indeks bi trebao biti što kraći jer neklasterirani indeksi sadrže kopije ključeva klasteriranog indeksa.
4. Ključevi klasteriranog indeksa bi trebali biti jedinstveni jer inače dolazi do povećavanja ključa zbog duplih vrijednosti.
5. Klasterirani indeks bi se trebao ubacivati po veličini, jer bi inače dolazilo do fragmentacije indeksa koja bi ga s vremenom usporavala.
6. Klasterirani indeks treba biti statičan jer bi također dolazilo do fragmentacije indeksa.
7. Preporučuje se postavljanje neklasteriranog indeksa na svaki stupac koji se koristi pri pretraživanju ili sortiranju ili je u pitanju strani ključ.

8. Veliki broj neklasteriranih indeksa usporava ubacivanje novih redova u tablicu.
9. Za ubrzavanje upita nad neklasteriranim indeksom kod kojeg se često traži manji broj dodatnih stupaca, ti se stupci mogu uključiti u indeks (kao uključeni stupci koji neće biti dio B-stabla indeksa), te se oni neće morati dodatno tražiti preko klasteriranog indeksa.
10. Numerički tipovi imaju najveće performanse, dok je razlika u performansama između različitih numeričkih tipova zanemariva.
11. Performanse tekstualnih indeksa ovise o stvarnoj veličini ključa u indeksu, a ne o deklariranoj, te je razlika između char i varchar iste veličine praktički zanemariva.
12. Binarno kodiranje teksta ima puno bolje performanse od teksta kodiranog kodnom stranicom nekog jezika.
13. Složeni indeksi usporavaju upite, te ukoliko je to moguće, preporučuje se ujediniti više manjih stupaca indeksa u jedan veći.

5. Reference

5. *References*

[1] Wikipedia, "Microsoft SQL Server," [Online]. Available: https://en.wikipedia.org/wiki/Microsoft_SQL_Server. [Accessed 25 March 2016].

[2] M. S. Rasmussen, "SQL Server Storage Internals 101," in Tribal SQL, Red Gate Books, 2013.

[3] Microsoft, "Clustered Index Structures," [Online]. Available: https://technet.microsoft.com/en-us/library/ms177443%28v=sql.105%29.aspx. [Accessed 4 March 2016].

[4] Microsoft, "Understanding Pages and Extents," [Online]. Available: https://technet.microsoft.com/en-us/library/ms190969%28v=sql.105%29.aspx. [Accessed 8 March 2016].

[5] M. Ufford, "Effective Clustered Indexes," [Online]. Available: https://www.simple-talk.com/sql/learn-sql-server/effective-clustered-indexes/. [Accessed 11 March 2016].

[6] Microsoft, "Nonclustered Index Structures," [Online]. Available: https://technet.microsoft.com/en-us/library/ms177484%28v=sql.105%29.aspx. [Accessed 11 March 2016].

[7] D. Comer, "The Ubiquitous B-Tree," Computing surveys, no. 11, pp. 123-137, 1979.

[8] Wikipedia, "B+ tree," [Online]. Available: https://en.wikipedia.org/wiki/B%2B_tree. [Accessed 3 March 2016].

[9] Wikipedia, "B-tree," [Online]. Available: https://en.wikipedia.org/wiki/B-tree. [Accessed 3 March 2016].

[10] Microsoft, "SET STATISTICS IO (Transact-SQL)," [Online]. Available: https://technet.microsoft.com/en-us/library/ms184361.aspx. [Accessed 15 March 2016].

[11] Microsoft, "SET STATISTICS TIME (Transact-SQL)," [Online]. Available: https://technet.microsoft.com/en-us/library/ms190287.aspx. [Accessed 15 March 2016].

[12] T. Chapman, "SQL Server Index Internals: A Deep Dive," [Online]. Available: https://www.youtube.com/watch?v=MpAKdy54Eqg. [Accessed 11 March 2015].

[13] P. Randal, "Inside the Storage Engine: Anatomy of a page," [Online]. Available: http://www.sqlskills.com/blogs/paul/inside-the-storage-engine-anatomy-of-a-page/. [Accessed 8 March 2016].

[14] R. Soukup and K. Delaney, "Optimizing Query Performance," [Online]. Available: https://technet.microsoft.com/en-us/library/cc917719.aspx#ECAA. [Accessed 15 March 2016].

AUTOR · AUTHOR

Željko Kovačević- nepromjenjena biografija nalazi se u časopisu Polytechnic & Design Vol. 2, No. 1, 2014.