****

**Evaluacija relacionih operatora kod MS SQL Server baze podataka**

**Mentor: Student:**

Prof. dr Aleksandar Stanimirović Draginja Anđelković, 1028

Niš, septembar 2021.

# Sadržaj

[1. Uvod 4](#_Toc83851802)

[2. Operator selekcije 6](#_Toc83851803)

[2.1. Nema indeksa, nesortirani podaci 6](#_Toc83851804)

[2.2. Nema indeksa, sortirani podaci 7](#_Toc83851805)

[2.3. Korišćenje indeksa B+ stabla 7](#_Toc83851806)

[2.4. Heš indeks, uslov jednakosti 8](#_Toc83851807)

[3. Generalni uslovi selekcije 10](#_Toc83851808)

[3.1. CNF i podudaranje indeksa 10](#_Toc83851809)

[3.2. Selekcija bez disjunkcije 10](#_Toc83851810)

[3.3. Selekcija sa disjunkcijom 11](#_Toc83851811)

[4. Operator projekcije 13](#_Toc83851812)

[4.1. Projekcija zasnovana na sortiranju 13](#_Toc83851813)

[4.2. Projekcija zasnovana na heširanju 14](#_Toc83851814)

[4.3. Projekcija: sortiranje ili heširanje 16](#_Toc83851815)

[4.4. Korišćenje indeksa za projekciju 17](#_Toc83851816)

[5. Spoj 18](#_Toc83851817)

[5.1. Spoj pomoću ugnježdenih petlji 19](#_Toc83851818)

[5.1.1. Spoj pomoću blok ugnježdenih petlji 20](#_Toc83851819)

[5.1.2. Spoj pomoću indeks ugnježdenih petlji 22](#_Toc83851820)

[5.2. Sortiranje-spajanje spoj 24](#_Toc83851821)

[5.2.1. Cena sortiranje-spajanje spoja 26](#_Toc83851822)

[5.2.2. Usavršavanje sortiranje-spajanje algoritma 27](#_Toc83851823)

[5.3. Heš spoj 28](#_Toc83851824)

[5.3.1. Zahtevi memorije i rukovanje prelivanjem 29](#_Toc83851825)

[5.3.2. Korišćenje dodatne memorije: Hibridni heš spoj 30](#_Toc83851826)

[5.3.3. Heš spoj ili spoj pomoću blok ugnježdenih petlji 31](#_Toc83851827)

[5.3.4. Heš spoj ili sortiranje-spajanje spoj 32](#_Toc83851828)

[6. Generalni uslovi spoja 33](#_Toc83851829)

[7. Operacije sa skupovima 34](#_Toc83851830)

[7.1. Pristup zasnovan na sortiranju za uniju i razliku 34](#_Toc83851831)

[7.2. Pristup zasnovan na heširanju za uniju i razliku 34](#_Toc83851832)

[8. Operacije agregacije 36](#_Toc83851833)

[8.1. Implementacija agregacionih operatora korišćenjem indeksa 37](#_Toc83851834)

[9. Uticaj baferovanja 38](#_Toc83851835)

[10. Zaključak 39](#_Toc83851836)

[11. Literatura 40](#_Toc83851837)

# Uvod

Relacioni operatori služe kao gradivni blokovi za obradu upita. Tehnike za njihovu implementaciju su proučavane od kada je napravljen prvi relacioni DBMS.

Upit se obično sastoji od nekoliko relacionih operatora. Odgovarajući plan fizičkog upita se sastoji od nekoliko fizičkih operatora. Uopšteno govoreći, fizički operator je implementacija relacionog operatora. Za svaki relacioni operator postoji nekoliko alternativnih algoritama (fizički operatori) za njegovu implementaciju i ne postoji univerzalno najbolji algoritam implementacije. Zbog toga je mudar odabir fizičkih operatora presudan za performanse upita. Koji je fizički operator najbolji zavisi od nekoliko faktora, kao što su veličina ulaznih relacija, postojeći poredak sortiranja ulaznih relacija, postojeći indeksi ili pogledi, politika zamene stranica bafera i veličina raspoloživog prostora u baferu itd. [3]

Ovaj rad govori o nekim osnovnim implementaijama relacionih operatora u tradicionalnim sistemima za upravljanje relacionim bazama podataka (engl. *Relational Database Management System* - RDBMS).

Algoritmi koji se koriste za implementaciju različitih relacionih operatora koriste zajedničke tehike:

* **Iteracija:** Obrada svih torki u ulaznoj relaciji iterativno, jedna za drugom. Ako je potrebno samo nekoliko polja iz svake torke, a postoji indeks čiji ključ sadrži sva polja, umesto da se pregledaju sve torke, mogu se skenirati indeksi ulazne relacije.
* **Ideksiranje:** Ako su specificirani uslovi selekcije ili spoja, koristi se indeks da bi se pregledale samo torke koje zadovoljavaju uslov.
* **Particionisanje:** Podelom torki na osnovu ključa za sortiranje, može se izvršiti dekompozicija operacije na kolekcije manje skupih operacija nad particijama. Sortiranje i heširanje su dve često korišćene tehnike particionisanja.

U nastavku ćemo govoriti o klasičnoj evaluaciji i imlementaciji relacionih operatora, jednog po jednog, i to o: selekciji, projekciji, spoju, Dekartovom proizvodu, standardnim operatorima teorije skupova (presek, unija, razlika) i grupisanju i operacijama agregacije.

U praktičnim primerima ovog rada korišćena je šema relacije baze podataka koja se sastoji od tri tabele, a koja se koristi za rezervaciju automobila u agenciji za iznajmljivanje automobila:

**Vozaci** (VID: integer, VIME: string, GODINE\_ISKUSTVA: integer, STAROST: integer)  
**Automobili** (AID: integer, MODEL: string, BOJA: string)  
**Rezervacije** (*VID: integer*, *AID: integer*, DAN: date, RIME: string)

Polja u tabeli *Vozaci* predstavljaju id vozača, ime vozača, godine vozačkog iskustva i starost vozača, respektivno. Id vozača je primarni ključ. Svaka torka relacije *Vozaci* je veličine 50 bajtova, ima 80 torki po stranici, a 500 stranica. U tabeli *Automobili* polja su id automobila, model i boja automobile, repektivno. Primarni ključ tabele je id automobila. Tabela *Rezervacije* sadrži id vozača, id automobila, dan za koji je izvršena rezervacija i ime osobe koja je izvršila rezervaciju, respektivno. Kompozitni primarni ključ ove relacije je kombinacija atributa *VID*, *AID* i *DAN*. *VID* je strani ključ relacije *Rezervacije*, koji ukazuje na primarni ključ *VID* tabele *Vozaci*, dok je *AID* strani ključ relacije *Rezervacije* koji ukazuje na primarni ključ *AID* tabele *Automobili*. Ime osobe u tabeli *Rezervacije*, atribut *RIME*, može se razlikovati od imena vozača sa id-jem *VID*. Svaka torka ove relacije je veličine 40 bajtova, ima 100 torki po stranici i 1000 stranica. U primerima iz ovog rada su od interesa i koriste se relacije *Vozaci* i *Rezervacije*.

Poglavlje 1 ovog rada predstavlja uvod u ubrađenu temu, Evaluacija relacionih operatora kod *Microsoft SQL Server* baze podataka.

U poglavlju 2 biće reči o algoritmima koji se koriste za implementaciju operacije selekcije.

Poglavlje 3 obrađuje generalne uslove selekcije.

U okviru poglavlja 4 obrađena je operacija projekcije. Objašnjene su dve najznačajnije tehnike za implementaciju projekcije, sortiranje i heširanje.

Poglavlje 5 ovog rada obrađuje implementaciju operacije spoja. Objašnjeni su različiti algoritmi za spajanje zasnovani na ugnježdenim petljama, sortiranje-spajanje algoritam, kao i algoritmi zasnovani na heširanju.

Generalni uslovi spoja obrađeni su u okviru poglavlja 6 ovog rada.

U okviru poglavlja 7 obrađena je implementacija operacija iz teorije skupova, i to preseka, unakrsnog proizvoda, unije i razlike.

Poglavlje 8 obrađuje implementaciju operacija agregacije i kombinaciju ovih operacija sa GROUP BY klauzulom za grupisanje.

U poglavlju 9 objašnjeno je kako baferovanje i strategija zamena stranica bafera utiče na implementaciju relacionih operatora.

Za svaki algoritam koji se koristi za implementaciju različitih relacionih operatora, prikazane su cene algoritama, na primeru šeme baze podataka za rezervaciju automobila.

# Operator selekcije

U ovom poglavlju se opisuju različiti algoritmi za procenu operatora selekcije. Za početak se razmatra upit prikazan na slici 1, koji ima uslov za izbor *RIME = 'Petar'.*

1.PNG

Slika : Jednostavan uslov selekcije

Ovaj upit se može izvršiti skeniranjem cele relacije, proverom uslova za svaku torku i dodavanjem torke rezultatu ako je uslov zadovoljen. Cena ovog pristupa je 1000 U/I, jer relacija *Rezervacije* sadrži 1000 stranica. Ako samo nekoliko torki ima *RIME = 'Petar',* ovaj pristup je skup jer ne koristi selekciju za smanjenje broja preuzetih torki. Za poboljšanje performansi ovog upita ključno je koristiti informacije iz uslova selekcije i koristiti indeks ako je odgovarajući indeks dostupan. Na primer, indeks B+ stabla na *RIME* mogao bi se koristiti za brži odgovor na ovaj upit, ali indeks na *AID*  ne bi bio od koristi.

U nastavku ovog poglavlja razmatraju se različite situacije u vezi sa organizacijom datoteka koje se koriste za relaciju i dostupnosti indeksa, i razmatraju se odgovarajući algoritmi za operaciju selekcije. Na početku će biti reči o jednostavnijoj operaciji selekcije oblika **op** , a zatim i o opštijoj selekciji. U smislu opštih tehnika za implementaciju, algoritmi za selekciju koriste ili ponavljanje (iteracije) ili indeksiranje.

## Nema indeksa, nesortirani podaci

S obzirom na selekciju oblika **op** , ako nema indeksa na i relacija R nije sortirana po vrednostima atributa , cela relacija mora da se skenira. Stoga je najselektivniji pristupni put skeniranje datoteke. Za svaku torku mora se testirati vrednost uslova **op** *value* i torka dodati u rezultat ako je uslov zadovoljen.

Cena ovakvog pristupa je M U/I operacija, gde je M broj stranica relacije R. U primeru koji razmatramo, gde relacija *Rezervacije* ima 1000 stranica, cena je 1000 U/I.

## Nema indeksa, sortirani podaci

S obzirom na selekciju oblika **op** , ako nema indeksa na *R.attr,* ali je relacija R fizički sortirana po vrednosti atributa *R.attr,* može se koristiti redosled sortiranja i koristiti binarno traženje za lociranje prve torke koja zadovoljava uslov selekcije. Zatim se uzimaju sve torke koje zadovoljavaju uslov selekcije kretanjem na ovoj lokaciji i skeniranjem relacije R sve dok uslov selekcije više nije zadovoljen. Metoda pristupa u ovom slučaju je skeniranje sortirane datoteke sa uslovima selekcije **op**

Na primer, pretpostavka je da je uslov selekcije *R.аttr* > 5, i da je relacija R sortirana po atributu attr u rastućem redosledu. Binarnim traženjem se locira pozicija u R koja odgovara vrednosti 5, a zatim se jednostavno skeniraju svi preostali zapisi.

Cena binarnog traženja je O(). Osim toga, postoje troškovi skeniranja za preuzimanje kvalifikovanih torki. Cena skeniranja zavisi od broja takvih torki i može varirati od 0 do M. U primeru sa slike 1, ukupna cena pretraživanja je , što je reda 10 U/I operacija.

U praksi je malo verovatno da će se relacija čuvati sortirana ako DBMS omogućava skladištenje zapisa podataka kao unose indeksiranih podataka. Ako je redosled zapisa podataka važan, bolji način za njegovo održavanje je korišćenje indeksa B+ stabla.

## Korišćenje indeksa B+ stabla

Ako je grupisani indeks B+ stabla dostupan na *R.attr,* najbolja strategija za uslov selekcije oblika **op**  u kojima operator **op** nije jednakost je upotreba indeksa. Ova strategija je takođe dobar pristupni put za selekciju sa uslovom jednakosti, iako bi heš indeks na *R.attr* atributu bio malo bolji pristup. Ako indeks B+ stabla nije grupisan, cena korišćenja indeksa zavisi od broja torki koje zadovoljavaju uslov selekcije.

Indeks se može koristiti na sledeći način. Pretražuje se stablo kako bi se pronašao prvi unos indeksa koji upućuje na kvalifikovanu torku R. Zatim se skenira stranica lista indeksa kako bi se dohvatili svi unosi u kojima vrednost ključa zadovoljava uslov selekcije. Za svaki od ovih unosa uzima se odgovarajuća torka R.

Troškovi identifikacije početne stranice za skeniranje obično su dva ili tri U/I. Troškovi skeniranja stranice na nivou lista za kvalifikovane unose podataka zavise od broja takvih unosa. Troškovi preuzimanja kvalifikovanih torki iz relacije R zavise od dva faktora:

* Broja odgovarajućih torki
* Da li je indeks grupisan

Ako je indeks grupisan, cena preuzimanja kvalifikovanih torki verovatno je samo jedan U/I stranice, pošto je verovatno da su sve takve torke sadržane na jednoj stranici. Ako indeks nije grupisan, svaki unos indeksa mogao bi ukazivati na kvalifikovane torke na drugoj stranici, a troškovi preuzimanja kvalifikovanih torki na jednostavan način mogli bi biti U/I jedne stranice po kvalifikovanoj torki. Broj U/I operacija za preuzimanje kvalifikovaih torki iz relacije R se može značajno smanjiti tako što se prvo sortiraju polja (u unosima podataka indeksa) prema njihovoj komponenti identifikatora stranice. Ovaj pristup osigurava da se, kada se unese stranica R, sve odgovarajuće torke na ovoj stranici preuzimaju jedna za drugom. Troškovi preuzimanja kvalifikovanih torki sada su broj stranica relacije R koje sadrže kvalifikovane torke.

Sada se razmatra selekcija oblika *RIME* < *'C%'* nad relacijom *Rezervacije*. Pretpostavljajući da su imena ravnomerno raspoređena u odnosu na početno slovo, radi jednostavnosti, procenjuje se da je otprilike 10% torki iz ove relacije u rezulatatu. Ovo je ukupno 10.000 torki ili 100 stranica. Ako imamo grupisani indeks B+ stabla na polju *RIME* relacije *Rezervacije,* kvalifikovane torke mogu da se preuzmu sa 100 U/I operacija, plus nekoliko U/I da bi se došlo od korena do odgovarajuće stranice lista, da bi se uopšte započelo skeniranje. Međutim, ako indeks nije grupisan, u najgorem slučaju bi bilo potrebno 10.000 U/I za preuzimanje, jer bi preuzimanje svake torke koštalo čitanja nove stranice. Ako se sortiraju torke relacije *Rezervacije* prema broju stranice, a zatim dohvate stranice relacije *Rezervacije*, izbegava se preuzimanje iste stranice više puta. Uprkos tome, torke koje treba preuzeti će verovato biti razbacane na mnogo više od 100 stranica, pa se može zaključiti da je upotreba negrupisanog indeksa za selekciju opsega potencijalno skupa. Čak bi jednostavno skeniranje cele relacije moglo biti jeftinije.

## Heš indeks, uslov jednakosti

Ako je heš indeks dostupan na atributu *R.attr* i operacija **op** je jednakost, najbolji način za implementaciju selekcije oblika **op**  je upotreba indeksa za preuzimanje kvalifikovanih torki.

Troškovi uključuju nekoliko (obično jedan ili dva) U/I za preuzimanje odgovarajuće stranice u indeksu, plus troškove preuzimanja kvalifikovanih torki iz relacije R. Troškovi preuzimanja kvalifikovanih torki iz R zavise od broja takvih torki i od toga da li je indeks grupisan. Budući da je operacija **op** jednakost, postoji tačno jedna kvalifikovana torka, ako je *R.attr* (kandidat) ključ za relaciju. Inače bismo mogli imati nekoliko torki sa istom vrednošću ovog atributa.

Razmatra se selekcija sa slike 1. Pretpostavka je da postoji negrupisan heš indeks na atributu *RIME*, da je na raspolaganju bafer sa 10 stranica i da su 100 rezervacija napravili ljudi po imenu *Petar*. Troškovi preuzimanja indeksirane stranice koja sadrži torke sa rezervacijama ljudi po imenu *Petar* su jedan ili dva U/I. Troškovi preuzimanja ovih 100 torki mogu varirati između 1 i 100, u zavisnosti od toga kako su ti zapisi raspoređeni po stranicama i redosleda po kojem se ti zapisi preuzimaju. Ako je ovih 100 zapisa sadržano u, recimo, nekih pet Reserves stranica, imamo samo pet dodatnih U/I, ako se zapisi sortiraju prema komponenti stranice. U suprotnom, moguće je da se unese jedna od ovih pet straica, zatim pogleda neka druga stranica i kada je prva ponovo potrebna, ona je izbačena iz bafera. Ovo može uzrokovati da ista stranica bude učitavana nekoliko puta. Situaciju komplikuje i činjenica da nekoliko korisnika i DBMS operacija dele isti bafer.

# Generalni uslovi selekcije

U prethodnom poglavlju razmatran je jednostavan uslov selekcije oblika **op** . Uopšteno, uslov selekcije je Booleova kombinacija[[1]](#footnote-2) uslova koji imaju oblik *atribut op konstanta* ili *atribut1 op atribut2*. Na primer, ako bi WHERE klauzula u upitu prikazanom na slici 1 sadržala složeniji uslov *R.RIME='Petar' AND R.AID=5*, ekvivalentni izraz algebre bi bio ʌ .

Sledeći deo ovog poglavlja definiše CNF. Prvo se razmatraju algoritmi za primenu uslova selekcije bez disjunkcije, a zatim sa disjunkcijom.

## CNF i podudaranje indeksa

Da bi se obradila operacija selekcije sa opštim uslovima izbora, prvo se uslov mora predstaviti u konjuktivnoj normalnoj formi (CNF), to jest, kao zbir konjunkata koji su povezani logičkim ʌ operatorom. Svaki konjunkt se sastoji od jednog ili više uslova povezanih logičkim v operatorom. [[2]](#footnote-3)

Kao primer, pretpostavlja se da postoji selekcija nad relacijom *Rezervacije* sa uslovom selekcije *(DAN* < *8/9/20 ʌ RIME* = *'Petar*') v *AID=5* v *VID=3.* Ovo se može prevesti u konjuktivnu normalnu formu kao *(DAN* < *8/9/20* v *AID=5* v *VID=3) ʌ (RIME =* *'Petar' v AID=5* v *VID=3).*

## Selekcija bez disjunkcije

Kada selekcija ne sadrži disjunkciju uslova, moraju se razmotriti dve mogućnosti evaluacije:

* Mogu se dohvatiti torke pomoću skeniranja datoteka ili jednog indeksa koji se podudara s nekim konjunktima (i za koji se procenjuje da je najselektivniji pristupni put) i primeniti svi neprimarni konjunkti u selekciji na svaku preuzetu torku. Ovaj pristup je veoma sličan načinu na koji se indeksi koriste za jednostavne uslove selekcije. Broj preuzetih torki zavisi od selektivnosti primarnih konjunkata u izboru, a preostali konjunkti samo umanjuju kardinalnost rezultata selekcije.
* Može se pokušati sa korišćenjem nekoliko indeksa. Ostatak ovog poglavlja posvećen je proučavanju ovog pristupa.

Ako se nekoliko indeksa koji sadrže unose podataka podudaraju sa konjunktima u selekciji, mogu se koristiti ti indeksi za izračunavanje skupova kandidatnih torki. Zatim se ovi skupovi redukuju, obično tako što se prvo sortiraju, a zatim dohvataju oni zapisi koji se nalaze u preseku. Ako su dodatni konjunkti prisutni u selekciji, mogu se primeniti kako bi se odbacile neke od kandidatskih torki iz rezultata.

Na primer, s obzirom na uslov *DAN* < *8/9/20 ʌ* *AID=5* ʌ *VID=3,* mogu se dohvatiti torke koje ispunjavaju uslov *DAN* < *8/9/20* koristeći indeks B+ stabla na *DAN*, preuzeti torke koje zadovoljavaju uslov *VID=3* pomoću heš indeksa na atributu *VID*, i naći presek ova dva skupa torki. Ako ove skupove sortiramo prema komponenti id stranice da bismo napravili presek, sporedna prednost je to što se torke u preseku dobijaju sortiranim redosledom po stranicama koje sadrže odgovarajuće torke, što osigurava da nećemo dohvatiti istu stranicu dva puta pri preuzimanju torki. Sada se mogu dohvatiti potrebne stranice relacije *Rezervacije* za preuzimanje torki i proveriti uslov *AID=5* kako bi se dobile torke koje ispunjavaju uslov *DAN* < *8/9/20 ʌ* *AID=5* ʌ *VID=3*.

## Selekcija sa disjunkcijom

Uzima se u obzir da je jedan od konjunkata u uslovu selekcije disjunkcija uslova. Ako makar jedan od ovih uslova zahteva skeniranje datoteke jer odgovarajući indeksi ili redosled sortiranja nisu dostupni, samo testiranje ove konjunkcije zahteva skeniranje datoteke. Na primer, pretpostavlja se da su jedini dostupni indeksi heš indeks na *RIME* i heš indeks na *VID,* i da uslov selekcije sadrži samo disjunkt (*DAN* < *8/9/20* v *RIME='Petar'*). Koristeći indeks na *RIME* mogu se dohvatiti torke koje zadovoljavaju uslov *RIME='Petar'*. Međutim, *DAN* < *8/9/20* zahteva skeniranje datoteke. Stoga se takođe može skenirati datoteka i proveriti uslov *RIME=* '*Petar'* za svaku preuzetu torku. Zbog toga je najselektivniji pristupni put u ovom primeru skeniranje datoteke.

S druge strane, ako je uslov selekcije (*DAN* < *8/9/20* v *RIME='Petar')* ʌ *VID=3,* indeks na *VID* odgovara konjunkciji *VID = 3*. Ovaj indeks se može koristiti za pronalaženje kvalifikovanih torki i primenu *DAN* < *8/9/20* v *RIME='Petar'* uslova samo na ove torke. Najbolji pristupni put u ovom primeru je indeks na atributu *VID* sa primarnim konjunktom *VID=3*.

Konačno, ako svaki izraz u disjunkciji ima indeks koji se podudara, mogu se dohvatiti torke kandidati pomoću indeksa, a zatim uzeti unija. Na primer, ako je uslov selekcije konjunktiv (*DAN* < *8/9/20* v *RIME='Petar'*), a postoje indeksi B+ stabla za *DAN* i *RIME,* mogu se dohvatiti sve torke takve da je *DAN <* *8/9/20* korišćenjem indeksa na *DAN*, i sve torke takve da je *RIME='Petar'* koristeći indeks na *RIME*, a zatim uzeti unija torki. Još bolji pristup je da se kvalifikovani zapisi sortiraju pre preuzimanja po broju stranice.

Većina trenutnih sistema ne rešava efikasno uslove selekcije sa disjunkcijom i fokusira se na optimizaciju izbora bez disjunkcije. Microsoft SQL Server razmatra upotrebu unije i bitmapa za rešavanje disjunktivih uslova.

# Operator projekcije

Za analizu operatora projekcije se razmatra upit prikazan na slici 2. Optimizator prevodi ovaj upit u izraz relacione algebre . Uopšteno gledano, operacija projekcije ima oblik . Za implementaciju projekcije, treba učiniti sledeće:

1. Ukloniti sve neželjene atribute (to jest one koji nisu navedeni u projekciji)
2. Ukloniti sve proizvedene duplikate.

2.PNG

Slika 2: Projekcija

Drugi korak je iz perspektive implementacije komplikovaniji. Postoje dva osnovna algoritma za uklanjanje duplikata, jedan zasnovan na sortiranju i jedan zasnovan na heširanju. U smislu opštih tehnika za implementaciju, oba algoritma su primeri particionisanja. Iako tehnika korišćenja indeksa za identifikaciju podskupa korisnih torki nije primenljiva na projekciju, algoritmi za sortiranje ili heširanje mogu se primeniti na unose podataka u indeks, umesto na zapise podataka.

## Projekcija zasnovana na sortiranju

Algoritam zasnovan na sortiranju ima sledeće korake:

1. Skeniranje relacije R i pravljenje skupa torki koji sadrže samo željene atribute
2. Sortiranje skupa torki korišćenjem kombinacije svih njegovih atributa kao ključa za sortiranje
3. Skeniranje sortiranog rezultata upoređivanjem susednih torki i odbacivanje duplikata.

Ako se u svakom koraku koriste privremene relacije, prvi korak košta M U/I za skeniranje relacije R, pri čemu je M broj stranica relacije R, a T U/I za pisanje privremene relacije, gde je T broj stranica privremene relacije. Tačna vrednost T zavisi od broja zadržanih polja i veličine ovih polja. Drugi korak košta O(TlogT). Poslednji korak košta T U/I. Ukupni troškovi su O(MlogM). Prvi i treći korak su jednostavni i relativno jeftini.

Sada se analizira projekcija nad relacijom *Rezervacije* prikazana na slici 2. Skeniranje relacije *Rezervacije* i pravljenje skupa torki koje sadrže samo željene atribute košta 1000 U/I. Ako se pretpostavi da je svaka torka u privremenoj relaciji kreiranoj u prvom koraku dugačka 10 bajtova, troškovi pisanja ove privremene relacije su 250 U/I. Pod pretpostavkom da je bafer veličine 20 stranica, privremena relacija se može sortirati u dva prolaza po ceni od 2\*2\*250 = 1000 U/I. Skreniranje koje se obavlja u trećem koraku košta dodatih 250 U/I. Ukupna cena je 2500 U/I.

Ovaj pristup se može poboljšati modifikovanjem algoritma za sortiranje kako bi se izvršila projekcija sa eliminacijom duplikata. Standardni algoritam za sortiranje uključuje nekoliko prolaza kroz relaciju. Prvi prolaz (prolaz 0) uključuje skeniranje zapisa koje treba sortirati da bi se proizveo početni skup interno sortiranih stranica privremene relacije. Nakon toga, sledi jedan ili više prolaza spajanja. Dve važne izmene algoritma za sortiraje ga prilagođavaju projekciji:

* Neželjeni atributi se mogu eliminisati tokom prvog prolaza (prolaza 0) sortiranja. Ako je dostupno B bafer stranica, možemo čitati i pisati B stranica relacije R, čime se dobija B interno sortiranih stranica bafera. Sa optimalnijom implementacijom može se zapisati u proseku 2B interno sortiraih stranica privremene relacije.
* Duplikati se mogu ukloniti tokom prolaza za spajanje. Ova izmena zapravo smanjuje cenu prolaza za spajanje jer u svakom prolazu ispisuje manje torki. Većina duplikata se eliminiše u prvom prolasku spajanja.

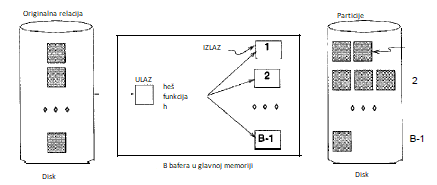
Ponovnim osvrtanjem na primer sa slike 2, vidi se da se u prvom prolazu skenira relacija *Rezervacije*, po ceni od 1000 U/I i ispisuje 250 stranica. Sa 20 bafer stranica, 250 stranica je ispisano u sedam interno sortiranih serija, gde je svaka, osim poslednje, dugačka oko 40 stranica. Ukupni troškovi su 1500 U/I, što je mnogo manje od troškova prvog pristupa koji se koristi za implementaciju projekcije.

## Projekcija zasnovana na heširanju

Ako je na raspolaganju relativno veliki broj bafer stranica B u odnosu na broj stranica relacije R, treba razmotriti pristup zasnovan na heširanju. Kod ovog pristupa postoje dve faze: podela na particije i uklanjanje duplikata.

U fazi particionisanja se koristi jedna stranica ulaznog bafera i B-1 stranica izlaznog bafera. Jedna po jedna stranica relacije R se učitava na stranicu ulaznog bafera. Ulazna stranica se obrađuje na sledeći način: Za svaku torku se odbacuju neželjeni atributi, a zatim se heš funkcija h primenjuje nad kombinacijom svih preostalih atributa. Funkcija h je izabrana tako da se torke ravnomerno raspoređuju na jednu od B-1 particija. Postoji jedna izlazna stranica po particiji. Nakon projekcije, torke se upisuju na izlaznu bafer stranicu na koju se hešira pomoću funkcije h.

Na kraju faze particionisanja imamo B-1 particiju, od kojih svaka sadrži kolekciju torki koje dele zajedničku heš vrednost i imaju samo željena polja. Faza podele prikazana je na slici 3.



Slika 3: Faza particionisanja projekcije zasnovane na heširanju

Garantovano je da dve torke koje pripadaju različitim particijama neće biti duplikati jer imaju različite heš vrednosti. Dakle, ako su dve torke duplikati, one se nalaze na istoj particiji. U fazi eliminisanja duplikata učitavaju se, jedna po jedna, B-1 particija, kako bi se eliminisali duplikati. Osnovna ideja je da se u memoriji kreira heš tabela dok se obrađuju torke, kako bi se otkrili duplikati.

Za svaku particiju kreiranu u prvoj fazi:

1. Čita se jedna po jedna stranica na particiji. Hešira se svaka torka primenom heš funkcije h2 (≠ *h*) nad kombinacijom svih polja, a zatim se umeće u heš tabelu u memoriji. Ako se nova torka hešira na istu vrednost kao neka postojeća torka, te dve torke se upoređuju kako bi se proverilo da li je nova torka duplikat. Čim se otkriju, duplikati se odbacuju.
2. Nakon što je čitava particija pročitana, torke se upisuju u heš tabelu, koja je bez duplikata, u datoteci rezultata. Zatim se heš tabela briše iz memorije i može se preći na sledeću particiju.

Treba imati na umu da je funkcija h2 namenjena za distribuciju torki u partiiji po mnogim segmentima radi minimiziranja kolizije[[3]](#footnote-4). Pošto sve torke u datoj particiji imaju istu vrednost h, h2 ne može biti isto što i h.

Strategija projekcije zasnovana na hešu neće dobro funkcionisati ako je veličina heš tabele za particiju, a koja je napravljena u fazi particionisanja, veća od broja dostupnih bafer stranica, B. Jedan od načina za rešavanje ovog problema prelivanja particije je rekurzivna primena tehnike projekcije zasnovane na heširanju za uklanjanje duplikata na svakoj particiji koja se preliva. To znači da se prenapunjena particija deli na podparticije, a zatim svaka podparticija učitava u memoriju kako bi se elimisali duplikati.

Ako se pretpostavi da h ravnomerno raspoređuje torke i da je broj stranica torki nakon projekcije, a pre uklanjanja duplikata T, svaka particija sadrži stranica. Ponovo, treba imati na umu da je broj particija B-1 jer se jedna od stranica bafera koristi za čitanje relacije tokom faze particionisanja. Veličina particije je stoga i veličina heš tabele za particiju je \* f, gde je f faktor neodređenosti (engl. *fudge factor*) koji se koristi za snimanje malog povećanja veličine između particije i heš tabele za particiju. Broj stranica bafera B mora biti veći od veličine particije \* f, kako bi se izbeglo prelivanje particije. To implicira da je potrebno približno *B* > bafer stranica.

U fazi particionisanja čita se relacija R, po ceni od M U/I. Takođe se ispisuju projektovane torke, ukupno T stranica, gde je T deo od M, u zavisnosti od polja koja su projektovana. Troškovi ove faze su M + T U/I. Troškovi heširaja su procesorski troškovi i oni se ne uzimaju u razmatranje. U fazi eliminisanja duplikata, obavlja se čitanje u svakoj particiji. Ukupan broj stranica na svim particijama je T. Takođe, nakon uklanjanja duplikata, popunjava se heš tabela koja je u memoriji za svaku particiju. Ova heš tabela je deo rezultata projekcije i zanemaruju se troškovi ispisivanja rezultujućih torki. Dakle, ukupni troškovi obe faze su M + 2T. U primeru sa slike 2, ukupan trošak projekcije je 1000 + 2\*250 = 1500 U/I.

## Projekcija: sortiranje ili heširanje

Pristup zasnovan na sortiranju je superiorniji od pristupa zasnovanog na heširanju ukoliko nakon projekcije postoji veliki broj duplikata ili ako je distribucija heš vrednosti veoma neujednačena. U ovom slučaju, neke particije bi mogle biti mnogo veće od proseka, a heš tabela za takvu particiju ne bi stala u memoriju tokom faze uklanjanja duplikata. Takođe, koristan sporedni efekat korišćenja pristupa zasnovanog na sortiranju je to što se dobija sortirani rezultat. Nadalje, budući da je spoljašnje sortiranje potrebno iz različitih razloga, većina sistema baza podataka ima pomoćni program za sortiranje, koji se može relativno lako implementirati. Iz ovih razloga, sortiranje je standardni pristup u implementaciji projekcije. Uklanjanje neželjenih atributa i uklanjanje duplikata su zasebni koraci u mnogim sistemima.

Uočava se da, ukoliko ima *B* > bafer stranica, gde je T veličina projektovane relacije pre uklanjanja duplikata, oba pristupa imaju iste U/I troškove. Sortiranje traje dva prolaza. U prvom prolazu se čita M stranica originalne relacije i ispisuje T stranica. U drugom prolazu se čita T stranica i ispisuje rezultat projekcije. Korišćenjem heširanja, u fazi particioisanja, čita se M stranica i zapisuju particije u vredosti od T stranica. U drugoj fazi se čita T stranica i ispisuje rezultat projekcije. Stoga razmatranja kao što su troškovi procesora, poželjnost sortiranog redosleda u rezultatu i iskrivljenost u raspodeli vrednosti utiču na izbor metode projekcije.

Microsoft SQL Server implementira algoritme zasnovane na heširanju i sortiranju.

## Korišćenje indeksa za projekciju

Ni pristup zasnovan na heširanju, ni sortiranje, ne koriste postojeće indekse. Postojeći indeks je koristan ako ključ uključuje sve atribute koji se zadržavaju u projekciji. U ovom slučaju, jednostavno se dohvataju ključne vrednosti iz indeksa, bez ikakvog pristupa stvarnoj relaciji, i primenjuju tehnike projekcije na ovaj, mnogo manji, skup stranica. Ova tehnika se naziva indeksno skeniranje. Ako posoji uređen indeks čiji ključ za pretragu uključuje željene atribute kao prefiks, algoritam se može dodatno poboljšati. Unosi podataka se u tom slučaju preuzimaju po redosledu, odbacuju se neželjena polja i upoređuju susedni unosi da bi se proverilo da li postoje duplikati.

# Spoj

Operacija spajanja (engl. *join*) objašnjena je oslanjanjem na upit sa slike 4.

4.PNG

Slika 4: Spoj

Ovaj upit se u relacionoj algebri može izraziti pomoću operacije spoja: R ⨝ S. Operacija spajanja, jedna od najkorisnijih operacija u relacionoj algebri, primarno je sredstvo za kombinovanje informacija iz dve ili više relacija.

Iako se spoj može definisati kao unakrsni proizvod praćen selekcijama i projekcijama, u praksi se spojevi pojavljuju mnogo češće od običnih unakrsnih proizvoda. Dalje, rezultat unakrsnog proizvoda je obično mnogo veći od rezultata spajanja, pa je veoma važno prepoznati spojeve i primeniti ih bez materijalizacije osnovnog unakrsnog proizvoda. Operaciji spajanja je zbog toga posvećena posebna pažnja.

U okviru ovog rada razmatra se nekoliko alternativnih tehnika za implementaciju spajanja. Započinje se raspravom o dva algoritma, jednostavne ugnežđene petlje i blok ugnežđene petlje, koji u suštini nabrajaju sve torke u unakrsnom proizvodu i odbacuju one koje ne zadovoljavaju uslove spajanja. Ovi algoritmi su primeri jednostavne iteracijske tehnike.

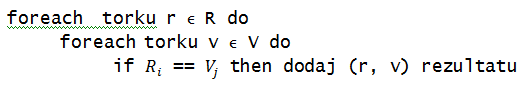
Preostali algoritmi spajanja izbegavaju računanje unakrsnog proizvoda. Oni su primeri tehnika indeksiranja i particionisanja. Intuitivno, ako je uslov spoja zapravo uslov jednakosti, može se zamisliti da torke iz dve relacije pripadaju particijama, tako da se samo torke na istoj particiji mogu spojiti jedna s drugom. Torke u particiji sadrže iste vrednosti u kolonama za spajanje. Spajanje pomoću indeks ugniježđene petlje skenira jednu od relacija i za svaku torku iz nje koristi indeks na koloni spoja druge relacije za lociranje torki na istoj particiji. Tako se samo podskup druge relacije upoređuje sa datim torkama iz prve relacije, a na uzima se u obzir ceo unakrsni proizvod. Poslednja dva algoritma, sortiranje-spajanje spoj (engl. *sort-merge join*) i heš spoj, takođe koriste uslove spajanja za particionisanje torki u relacijama, čime se spajaju i upoređuju samo torke na istoj particiji dok se računa spoj, ali se ne oslanjaju na prethodno postojeći indeks. Umesto toga, oni ili sortiraju ili heširaju relacije koje se spajaju da bi se postiglo particionisanje.

Valja napomenuti da Microsoft SQL Server podržava blok ugnežđene petlje, indeks ugnežđene petlje, heš spoj i tehniku koja se naziva heš timovi.

U sledećim odeljcima ovog poglavlja se razmatra spajanje dve relacije R i V, sa uslovom spajanja = . Pretpostavlja se M stranica u relaciji R sa torki po stranici i N stranica u relaciji V sa torki po stranici. U predstavljanju algoritama koriste se oznake R i V, a za posebne primere relacije *Rezervacije*  i *Vozaci*.

## Spoj pomoću ugnježdenih petlji

Najjednostavniji algoritam za spajanje je algoritam koji se zasniva na ugnježdenim petljama orijentisan ka torkama (engl. *tuple-at-a-time*). Ovaj algoritam je prikazan na slici 5. Skenira se spoljašnja relacija R i za svaku torku r ϵ R skenira se cela unutrašnja relacija V. Cena skeniranja relacije R je M U/I. Relacija V se skenira ukupno M puta, a svako skeniranje košta N U/I. Dakle, ukupni troškovi su M + \* M \* N.



Slika : Spoj pomoću jednostavnih ugnježdenih petlji

Relacija *Rezervacije* označena je slovom R, a relacija *Vozaci* slovom V. Vrednost M je 1.000, je 100, a N je 500. Troškovi spajanja pomoću jednostavnih ugnežđenih petlji su 1000 + 100 \* 1000 \* 500 stranica U/I. Takođe, postoje i troškovi ispisivanja rezultata, ali se oni mogu zanemariti. Vidimo da je cena ovakvog pristupa ogromna: 1000 + (5 \* ) U/I. Treba imati na umu da svaki U/I košta oko 10 ms na trenutnom hardveru, što znači da bi ovo spajanje trajalo oko 140 sati!

Jednostavno poboljšanje ovog pristupa bilo bi spajanje orijentisano ka stranici (engl. *page-at-a-time*) umesto spajanja orijentiranog ka torkama. Za svaku stranicu relacije R, može se dohvatiti svaka stranica relacije V i ispisati torka (r, v) za sve kvalifikacione torke r ϵ R-stranica i v ϵ V-stranica. Na ovaj način, cena je M za skeniranje relacije R, kao i ranije. Međutim,V se skenira samo M puta, pa je ukupna cena M + M \* N. U primeru relacija *Rezervacije* i *Vozaci*, troškovi su smanjeni na 1000 + 1000 \* 500 = 501.000 U/I i trajali bi oko 1,4 sata. Ovo dramatično poboljšanje naglašava važnost operacija orijenisanih ka stranicama za minimiziranje U/I operacija diska.

Iz ovih formula troškova jasno se vidi da bi za spoljašnju relaciju trebalo izabrati manju od dve relacije koje se spajaju. Za operaciju spoja se može reći da je komutativna, sve dok se vodi računa o nazivima polja. Izbor veće relacije, relacije R, za spoljašnju relaciju ne menja značajno troškove. Ako izaberemo manju relaciju, *Vozaci*, kao spoljašnju, cena algoritma po stranici je 500 + 500 \* 1000 = 500.500 U/I, što je samo neznatno bolje od cene jednostavne ugnežđene petlje orijentisane ka stranici.

### Spoj pomoću blok ugnježdenih petlji

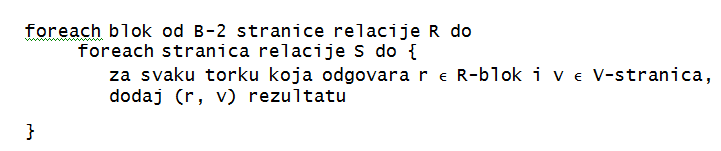
Jednostavan algoritam za spajanje pomoću ugnežđenih petlji ne koristi efikasno stranice bafera. Ukoliko je na raspolaganju dovoljno memorije za držanje manje relacije, recimo, R, sa preostale najmanje dve dodatne stranice bafera, može se učitati manja relacija i koristiti jedna od dodatnih memorijskih stranica za skeniranje veće relacije V. Za svaku torku v ϵ V, proverava se R i ispisuje torka (r', v). Druga stranica dodatnog bafera se koristi kao izlazni bafer. Svaka relacija se skenira samo jednom, pa su ukupni U/I troškovi M + N, što je optimalno.

Ako je na raspolaganju dovoljno memorije, važno poboljšanje je izgradnja memorijske heš tabele za manju relaciju R. Troškovi U/I su i dalje M + N, ali su troškovi CPU -a tipično mnogo niži sa usavršavanjem heš tabele.

Šta ako je na raspolaganju premalo memorije da se zadrži čitava manja relacija? Prethodna ideja se može generalizovati razbijanjem relacije R u blokove koji se mogu uklopiti u dostupne stranice bafera i skeniranjem svih V za svaki blok R. R je spoljnja relacija, jer se skenira samo jednom, a V je unutrašnji relacija, jer se skenira više puta. Ako je na raspolaganju B memorijskih stranica, na B-2 stranice se može učitati spoljna relacija R i skenirati unutrašnja relacija V korišćenjem jedne od dve preostale stranice. Mogu se ispisati nizovi (r', v), gde je r ϵ R-blok, v ϵ V-stranica i = , korišćenjem poslednje stranice bafera za izlaz.

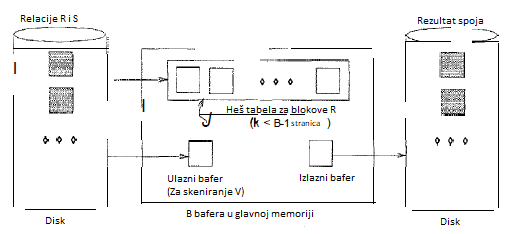
Efikasan način za pronalaženje odgovarajućih parova, to jest torki koje zadovoljavaju uslov spoja = , je izgradnja heš tabele u glavnoj memoriji za blok R. Pošto heš tabela za skup torki zauzima malo više prostora osim prostora neophodnog za čuvanje samih torki, pravljenje heš tabele uključuje kompromis. Efikasna veličina bloka R, u smislu broja torki po bloku, je smanjena. Izgradnja heš tabele je tehnika koja vredi truda.

Algoritam za spoj pomoću blok ugnježdenih petlji prikazan je na slici 6.



Slika : Spoj pomoću blok ugnježdenih petlji

Cena ove strategije je M U/I za čitanje relacije R, koja se skenira samo jednom. V se skenira ukupno puta, zanemarujući dodatni prostor potreban po stranici zbog heš tabele u memoriji i svako skeniranje košta N U/I. Ukupni troškovi su stoga *M + N \* .*

**

Slika 7: Korišćenje bafera u spoju pomoću blok ugnježdenih petlji

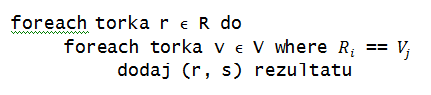
Prilikom spoja relacija *Rezervacije* i *Vozaci*, ukoliko se realcija *Rezervacije* izabere kao spoljašnja relacija R i pretpostavi se da je na raspolaganju dovoljno veliki bafer da se drži memorijska heš tabela za 100 stranica relacije *Rezervacije*, mora se skenirati relacija *Rezervacije*, po ceni od 1000 U/I, zatim za svaki blok stranica relacije *Rezervacije* na 10 stranica se mora skenirati relacija *Vozaci*. Zbog toga se vrši 10 skeniranja relacije *Vozaci*, a svako skeniranje košta 500 U/I. Ukupni troškovi su 1000 + 10 \* 500 = 6000 U/I. Ukoliko bi na raspolaganju bilo dovoljno bafer stranica za 90 stranica relacije *Rezervacije*, bilo bi neophodno skenirati relaciju *Vozaci* = 12 puta, a ukupni troškovi bi bili 1000 + 12 \* 500 = 7000 U/I.

Ako se relacija *Vozaci* izabere kao spoljašnja relacija umesto relacije *Rezervacije*, skeniranje relacije *Vozaci* košta 500 U/I. U tom slučaju bi se relacija *Rezervacije* skenirala = 5 puta. Ukupni troškovi bili bi 500 + 5 \* 1.000 = 5500 U/I. Ako umesto toga postoji dovoljno bafer stranica za 90 stranica relacije *Vozaci*, relacija *Rezervacije* bi se skenirala ukupno = 6 puta. Ukupni trošak u ovom slučaju bio bi 500 + 6 \* 1000 = 6500 U/I. Algoritmu za spajanje pomoću ugnježđenih petlji potrebno je nešto više od jednog minuta na primeru koji je ovde korišćen, pretpostavljajući 10 ms po U/I kao i ranije.

Vredi napomenti i da, ukoliko se uzme u obzir i učestalost promašaja stranica u baferu, dolazi do fundamentalne promene u načinu dodeljivanja bafera kod spoja pomoću blok ugnježdenih petlji. Umesto da se koristi samo jedna stranica bafera za unutrašnju relaciju, najbolji pristup je ravnomerno podeliti bafer između dve relacije. Ova alokacija rezultira većim brojem prolaska preko unutrašnje relacije, što dovodi do češćeg preuzimanja stranica. Međutim, vreme provedeno na traženju stranica dramatično se smanjuje. Takođe se može koristiti tehnika dvostrukog baferovanja, ali ona prevazilazi obim ovog rada.

### Spoj pomoću indeks ugnježdenih petlji

Ako postoji indeks na atributima spoja na jednoj od relacija, može se iskoristiti indeks tako što će se indeksirana relacija učiniti unutrašnjom. Ukoliko postoji odgovarajući indeks na relaciji V, slika 8 opisuje algoritam za spajanje pomoću indeks ugnježdene petlje.



Slika : Spoj pomoću indeksnih ugnježdenih petlji

Za svaku torku r ϵ R, koristi se indeks za dohvatanje odgovarajućih torki iz relacije V. Intuitivno, r se upoređuje samo sa torkama iz V koje su na istoj particiji, po tome što imaju istu vrednost u uslovu spoja. Za razliku od drugih algoritama za spoj pomoću ugnežđenih petlji, algoritam za spajanje pomoću indeksnih ugnežđenih petlji ne računa unakrsni proizvod relacija R i V. Cena skeniranja relacije R je M, kao i ranije. Troškovi preuzimanja odgovarajućih V torki zavise od vrste indeksa i broja odgovarajućih torki. Za svaku torku iz relacije R cena je sledeća:

1. Ako je indeks na V indeks B+ stabla, cena pronalaženja odgovarajućeg lista obično je 2-4 U/I. Ako je indeks heš indeks, cena pronalaženja odgovarajuće torke je 1-2 U/I.
2. Kada se pronađe odgovarajući list ili torka, troškovi preuzimanja odgovarajućih torki iz relacije V zavise od toga da li je indeks grupisan. Ako jeste, cena po spoljnoj torki r ϵ R je tipično samo još jedan U/I. Ako nije grupisan, cena bi mogla biti jedan U/I po odgovarajućoj V torki, budući da bi svaka od njih u najgorem slučaju mogla biti na drugoj stranici.

Kao primer, pretpostavlja se da postoji indeks zasnovan na hešu na atributu *VID* relacije *Vozaci* i da je u proseku potrebno oko 1,2 U/I[[4]](#footnote-5) da se preuzme odgovarajuća stranica indeksa. Budući da je *VID*  ključ za *Vozaci*, postoji najviše jedna odgovarajuću torka. Zaista, *VID* u relaciji *Rezervacije* je strani ključ koji se odnosi na relaciju *Vozaci*, pa postoji tačno jedna odgovarajuća *Vozaci* torka za svaku torku iz *Rezervacije*. Ako se uzmu u obzir troškovi skeniranja relacije *Rezervacije* i korišćenja indeksa za pronalaženje odgovarajuće troke relacije *Vozaci* za svaku torku iz relacije *Rezervacije*, troškovi skeniranja relacije *Rezervacije* su 1000 U/I. U ovoj relaciji ima 100 \* 1000 torki. Za svaku od ovih torki, preuzimanje indeksne stranice koja sadrži odgovarajuće torke relacije *Vozaci* košta u proseku 1,2 U/I. Pored toga, stranica relacije *Vozaci* koja sadrži kvalifikovanu torku mora da se preuzme. Prema tome, troškovi su 100000 \* (1 + 1.2) U/I za preuzimanje odgovarajućih torki relacije *Vozaci*, što je ukupno 221.000 U/I.

Kao još jedan primer, pretpostavlja se da postoji indeks zasnovan na hešu na *VID* atributu relacije *Rezervacije*. Sada se relacija *Vozaci* može skenirati po ceni 500 U/I, i za svaku torku se upotrebljava indeks za preuzimanje odgovarajućih torki relacije *Rezervacije*. Ukupno ima 80 \* 500 torki relacije *Vozaci*, a svaka torka se može podudarati sa 0 ili više torki relacije *Rezervacije*. Za svaku torku relacije *Vozaci* po ceni od 1,2 U/I se može dohvatiti indeksna stranica koja sadrži odgovarajuće torke iz relacije *Rezervacije*, pod pretpostavkom da imamo najviše jednu takvu indeksnu stranicu, što je razumna pretpostavka. Ukupni dosadašnji troškovi su 500 + 40.000 \* 1.2 = 48.500 U/I.

Osim toga, postoje troškovi preuzimanja odgovarajućih torki relacije *Rezervacije*. S obzirom da ima 100.000 torki u relaciji *Rezervacije* za 40.000 torki u relaciji *Vozaci*, a pod pretpostavkom uniformne distribucije, može se proceniti da se svaka torka iz relacije *Vozaci* podudara sa 2.5 torki iz relacije *Rezervacije* u proseku. Ako je indeks u relaciji *Rezervacije* grupisan, a ove podudarne torke se obično nalaze na istoj stranici relacije *Rezervacije* za određenu torku relacije *Vozaci*, troškovi njihovog preuzimanja su samo jedan U/I po torki iz relacije *Vozaci*, što dodaje do 40.000 dodatnih U/I. Ako indeks nije grupisan, svaka odgovarajuća torka iz relacije *Rezervacije* može se nalaziti na drugoj stranici, što dovodi do ukupno 2,5 \* 40.000 U/I za preuzimanje kvalifikovanih torki. Prema tome, ukupni troškovi mogu varirati od 48.500 + 40.000 = 88.500 do 48.500 + 100.000 = 148.500 U/I. Pretpostavljajući 10 ms po U/I, ovo bi trajalo oko 15 do 25 minuta.

Dakle, čak i sa neklasterisanim indeksom, ako je broj odgovarajućih unutrašnjih torki za svaku spoljašnju torku mali, cena algoritma za spajanje pomoću indeks ugnežđenih petlji verovatno će biti mnogo manja od cene jednostavnog spoja pomoću ugnežđenih petlji.

## Sortiranje-spajanje spoj

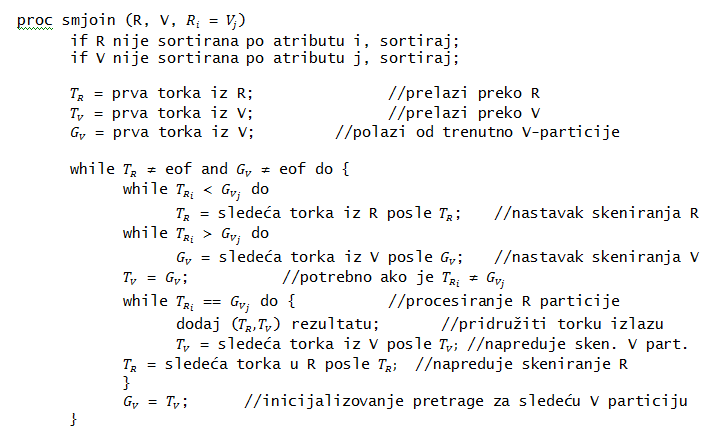
Osnovna ideja iza algoritma za spajanje sortiranje-spajanje (engl. *sort-merge*) je sortiranje obe relacije po atributu spoja, a zatim traženje kvalifikacionih torki r *ϵ* R i v *ϵ* V spajanjem dve relacije. Korak sortiranja grupiše sve torke koje imaju istu vrednost u koloni spoja i na taj način olakšava identifikaciju particija ili grupa torki sa istom vrednošću u koloni spoja. Ova particija se iskorišćava za upoređivanje torki iz relacije R na particiji samo sa torkama iz relacije V na istoj particiji (umesto sa svim torkama iz V), čime se izbegava određivanje unakrsnog proizvoda relacija R i V. Treba imati na umu da ovaj pristup zasnovan na particiji funkcioniše samo kod spojeva sa uslovom jednakosti.

Ako je relacija već sortirana po atributu spoja, ne mora se ponovo sortirati. U koraku spajanja se skeniraju relacije R i V i traže kvalifikacione torke, to jest torke u realciji R i u relaciji V tako da je = . Dva skeniranja započinju na prvoj torci u svakoj relaciji. Skeniranje napreduje kroz R sve dok je trenutna R torka manja od trenutne V torke, s obzirom na vrednosti u atributu spoja. Slično tome, skeniranje V napreduje sve dok je trenutna V torka manja od trenutne R torke. Naizmenično se menjaju takve torke sve dok se ne pronađe torka u R i u V koja zadovoljava uslov = .

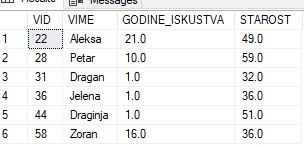
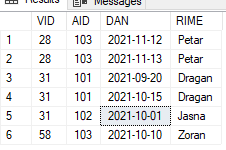
Kada se pronađu torke i takve da je = , spojena torka se ispisuje. U stvari, može se desiti da postoji nekoliko R torki i nekoliko V torki sa istom vrednošću u atributima spoja kao i trenutne torke i . Ove torke se nazivaju trenutna R particija i trenutna V particija. Za svaku torku r u trenutnoj R particiji, skeniraju se sve torke v na trenutnoj V particiji i dobijaju torke spoja (r, v). Zatim se nastavlja sa skeniranjem R i V, počevši od prvih torki koji slede particije torki koje su upravo obrađene.

Algoritam za sortiranje i spajanje prikazan je na slici 9. Promenljivama , i dodeljuju se samo vrednosti torki i koristi posebna vrednost eof da se označi da nema više torki u relaciji koja se skenira. Indeksi identifikuju polja, na primer, označava i-to polje torke . Ako ima vrednost eof, svako poređenje koje uključuje je definisano tako da se oceni kao netačno. Ilustracije algoritma sortiranje-mešanje na primeru relacija *Vozaci* i *Rezervacije* prikazane su na slikama 10 i 11, pri čemu je uslov spajanja jednakost *VID* atributa.

Ove dve relacije su već sortirane po atrubutu *VID*, a faza spajanja algoritma počinje skeniranjem pozicioniranim na prvoj torci svake instance relacije. Skeniranje relacije *Vozaci* napreduje, jer je njegova vrednost atributa *VID* u primeru jednaka 22, manja od vrednosti *VID* za relaciju *Rezervacije*, koja je 28. Druga torka iz relacije *Vozaci* ima *VID = 28*, što je jednako vrednosti *VID* atributa za trenutnu torku relacije *Rezervacije*. Stoga se ispituju rezultujuće torke za svaki par torki, jedna iz relacije *Vozaci*  i jedna iz relacije *Rezervacije*, na trenutnoj particiji, to jest particiji sa *VID = 28*. S obzirom da imamo samo jednu *Vozaci* torku sa *VID = 28* i dve takve torke iz relacije *Rezervacije*, u rezultat ulaze dve torke. Nakon ovog koraka, skeniranje *Vozaci* relacije se pozicionira na prvu torku nakon particije sa *VID = 28*, koja ima *VID = 31.* Slično, pozicionira se skeniranje *Rezervacije* relacije na prvu torku sa *VID = 31*. Pošto ove dve torke imaju iste *VID* vrednosti, pronađena je sledeća podudarna particija i treba da se ispišu nizovi rezultata generisani od ove particije, pri čemu postoje tri takve torke. Nakon toga, skeniranje relacije *Vozaci* se postavlja za torke sa *VID = 36*, a skeniranje relacije *Rezervacije* se postavlja na torke sa *VID = 58*. Ostatak faze spajanja se odvija na sličan način.



Slika : Sortiranje-spajanje spoj

Slika : Instanca relacije Vozaci Slika : Instanca relacije Rezervacije

Uopšteno, particija torki u drugoj relaciji se mora skenirati onoliko često koliko i broj torki u odgovarajućoj particiji u prvoj relaciji. Prva relacija u primeru, *Vozaci*, ima samo po jednu torku na svakoj particiji. Ovo nije slučajnost, već posledica činjenice da je *VID* ključ relacije. Nasuprot tome, ako se uslov spoja promeni u *VIME =RIME*, obe relacije sadrže više od jedne torke na particiji sa *VIME = RIME = 'Dragan'.* Torke *sa RIME = 'Dragan'* u relaciji *Rezervacije* moraju se skenirati za svaku *Vozaci*  torku sa *VIME = 'Dragan'*.

### Cena sortiranje-spajanje spoja

Cena sortiranja relacije R je O (MlogM), a cena sortiranja relacije V je O (NlogN). Cena faze spajanja je M + N ako se nijedna V particija ne skenira više puta, ili se potrebne stranice nađu u baferu nakon prvog prolaza. Ovaj pristup je posebno atraktivan ako je barem jedna relacija već sortirana na atributu spoja ili ima klasterisani indeks na atributu spoja.

Ako se razmatra spajanje relacija *Rezervacije* i *Vozaci*, a pod pretpostavkom da imamo 100 bafer stranica, otprilike isti broj koji je pretpostavljeno da je dostupan u raspravi o spoju pomoću blok ugnežđenih petlji iz poglavlja 5.1.1, relacija *Rezervacije* se može sortirati u samo dva prolaza. Prvi prolaz proizvodi 10 interno sortiranih serija od po 100 stranica. Drugi prolaz spaja ovih 10 serija kako bi proizveo sortiranu relaciju. Pošto se relacija *Rezervacije* čita i piše u svakom prolazu, troškovi sortiranja su 2 \* 2 \* 1000 = 4000 U/I. Slično tome, možemo sortirati relaciju *Vozaci* u dva prolaza, po ceni od 2 \* 2 \* 500 = 2000 U/I. Osim toga, druga faza algoritma sortiranja i spajanja zahteva dodatno skeniranje obe relacije. Tako ukupni trošak iznosi 4000 + 2000 + 1000 + 500 = 7500 U/I, što je slično ceni algoritma pomoću ugnežđenih blok petlji.

Ako se pretpostavi da je na raspolaganju samo 35 bafer stranica, i dalje se relacije *Rezervacije* i *Vozaci* mogu sortirati u dva prolaza, a troškovi algoritma za sortiranje i spajanje ostaju 7500 U/I. Međutim, cena algoritma za spajanje pomoću ugnežđenih blok petlji je veća od 15.000 U/I. S druge strane, ako je na raspolaganju 300 stranica bafera, troškovi spajanja sortiranjem-spajanjem ostaju na 7500 U/I, dok troškovi spajanja ugniježđenih blok petlji padaju na 2500 U/I.

Treba napomenuti da je višestruko skeniranje particije druge relacije potencijalno skupo. U primeru koji se razmatra, ako je broj torki relacije *Rezervacije* na više puta skeniranoj particiji mali, recimo, samo nekoliko stranica, verovatnoća da se cela particija pronađe u baferu pri ponovljenim skeniranjima je veoma velika, a U/I troškovi ostaju u suštini isti kao i za jedno skeniranje. Međutim, ako se na određenoj particiji nalazi mnogo stranica torki iz relacije *Rezervacije*, prva stranica takve particije možda više neće biti u baferu kada je drugi put potrebna. U ovom slučaju, U/I troškovi mogu biti visoki koliko i broj stranica na particiji relacije *Rezervacije* puta broj torki u odgovarajućoj particiji relacije *Vozaci*.

U najgorem scenariju, faza spajanja mogla bi zahtijevati da se pročita kompletna druga relacija za svaku torku u prvoj relaciji, a broj U/I je O (M \* N) U/I. Do ovog scenarija dolazi kada sve torke u obe relacije sadrže istu vrednost u atributu spoja, što je vrlo malo verovatno.

U praksi, U/I troškovi faze spajanja su obično samo jedno skeniranje svake relacije. Pojedinačno skeniranje može biti zagarantovano ako barem jedan od uključenih relacija nema duplikate u atributu spoja.

### Usavršavanje sortiranje-spajanje algoritma

U primeru koji se razmatra pretpostavljeno je da se dve relacije prvo sortiraju, a zatim spajaju u različitom prolazu. Moguće je poboljšati algoritam sortiranja-spajanja kombinovanjem faze sortiranja sa fazom spajanja. Prvo, proizvode se sortirani redovi veličine B i za relaciju R i za relaciju V. Ako je *B* > , gde je L veličina veće relacije, broj izvođenja po relaciji je manji od . Ako je broj raspoloživih bafer stranica za fazu spajanja najmanje , to jest, više od ukupnog broja izvođenja za R i V, dodeljuje se jedna bafer stranica za svaki niz R i jedna za svaki niz V. Zatim se R serije spajaju sa V serijama i spoje rezultujući tokovi R i V. Uslov spoja se primenjuje dok se spajaju R i V tokovi i odbacuju torke iz unakrsnog proizvoda koje ne ispunjavaju uslov spoja.

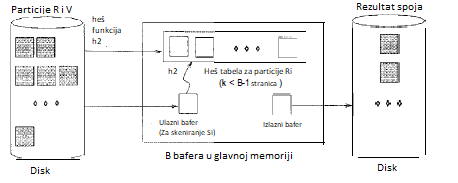
Nažalost, ova ideja povećava broj potrebnih bafer stranica do *2* . Međutim, postoji tehika za proizvođenje sortiranih serija veličine približno 2 \* B za obe relacije R i V. Prema tome, imaće se manje od izvođenja za svaku relaciju, s obzirom na pretpostavku da je *B* > . Dakle, ukupan broj izvođenja je manji od , to jest manji od B, i mogu se kombinovati faze spajanja bez potrebe za dodatnim baferima.

Ovaj pristup omogućava izvođenje sortiranje-spajanje spoja po ceni čitanja i pisanja R i V u prvom prolasku i čitanja R i V u drugom prolazu. Ukupni trošak je stoga 3 \* (M + N). U našem primeru, cena se smanjuje sa 7500 na 4500 U/I.

Takođe, tehnika dvostrukog baferovanja se može koristiti za ubrzavanje faze spajanja, kao i faze sortiranja relacija koje se spajaju.

## Heš spoj

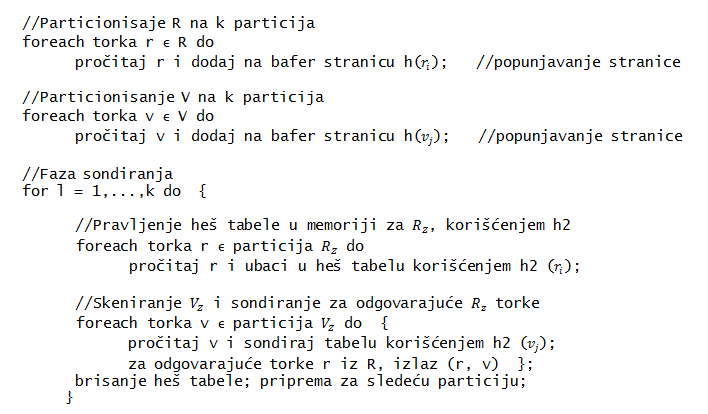
Algoritam heš spoja, poput algoritma sortiranje-spajanje, identifikuje particije u R i V u fazi particionisanja i, u sledećoj fazi sondiranja (engl. *probing*), upoređuje torke u R particiji samo sa torkama u odgovarajućoj V particiji radi testiranja uslova jednakosti spoja. Za razliku od sortiranje-spajanje spoja, heš spoj koristi heširanje za identifikaciju particija umesto sortiranja. Faza particioniranja heš spoja je slična particionisanju u heš projekciji i ilustrovana je na slici 3. Faza sondiranja ilustrovana je na slici 12.



Slika 12: Faza sondiranja heš spoja

Ideja je da se heširaju obe relacije po atributu spoja, koristeći istu heš funkciju h. Ako se svaka relacija hešira, idealno ravnomerno, u k particija, sigurno je da se R torke u particiji i mogu spojiti samo sa V torkama u istoj particiji i. Ovo zapažanje može se koristiti s dobrim efektom. Mogu se čitati particije manje relacije R i skenirati samo odgovarajuća particija V za podudarnosti. Više nikada ne mora da se razmatraju ove R i V torke. Dakle, kada su R i V podeljeni, može se izvršiti spajanje čitanjem R i V samo jednom, pod uslovom da je na raspolaganju dovoljno memorije za držanje svih torki u bilo kojoj particiji R.

U praksi se u memoriji pravi heš tabela za R particiju, korišćenjem heš funkcije h2 koja se razlikuje od h, budući da je h2 namenjena distribuciji torki na particiji zasnovanoj na h, kako bi se smanjili troškovi CPU-a. Potrebno je dovoljno memorije za držanje ove heš tabele, koja je malo veća od same R particije. Algoritam heš spoja prikazan je na slici 13. Ako se uzme u obzir cena algoritma heš spajanja, u fazi particioniranja jednom se mora skenirati i R i V i jednom ih treba zapisati. Troškovi ove faze su stoga 2 \* (M + N). U drugoj fazi, skenira se svaka particija jednom, pretpostavljajući da se particija ne preliva, po ceni od M + N U/I. Ukupni troškovi su stoga 3 \* (M +N), s obzirom na pretpostavku da se svaka particija uklapa u memoriju u drugoj fazi. Na primeru spajanja relacija *Rezervacije* i *Vozaci*, ukupni trošak je 3 \* (500 + 1000) = 4500 U/I, a pod pretpostavkom 10 ms po U/I, heširanje traje manje od jedog minuta. Ako se ovo uporedi sa spojem pomoću jednostavnih ugnežđenih petlji koje bi za isti primer trajale oko 140 sati, vidi se važnost korišćenja dobrog algoritma spajanja.



Slika 13: Heš spoj

### Zahtevi memorije i rukovanje prelivanjem

Da bi se povećala šansa da se data particija uklopi u dostupnu memoriju u fazi ispitivanja, mora se minimizirati veličina particije povećanjem broja particija. U fazi particionisanja, za podelu relacije R (a slično važi i za relaciju V) na k particija, potrebno je najmanje k izlaznih bafera i jedan ulazni bafer. Stoga, s obzirom na stranice B bafera, maksimalni broj particija je k = B - 1. Pod pretpostavkom da su particije jednake veličine, to znači da je veličina svake R particije , pri čemu je M je broj stranica relacije R. Broj stranica u memoriji heš tabele izgrađene tokom faze ispitivanja za particiju je + 2, gde je f faktor koji se koristi za čuvanje malih povećanja veličine između particije i heš tabele za particiju.

Tokom faze ispitivanja, pored heš tabele za particiju relacije R, potrebna je stranica bafera za skeniranje particije relacije V i izlazni bafer. Zbog toga nam potrebno *B* > + 2. Potrebno je približno *B* > \* M da bi algoritam heš spoja radio dobro.

Budući da će particije relacije R verovatno biti približno iste veličine, ali neće biti identične, najveća particija je nešto veća od , a broj potrebnih stranica bafera je nešto veći od *B* > \* M*.* Takođe postoji rizik da, ako heš funkcija h ne podeli relaciju R ravnomerno, heš tabela za jednu ili više particija relacije R možda neće stati u memoriju tokom faze ispitivanja. Ova situacija može značajno umanjiti performanse.

Kao što je već napomenuto u kontekstu projekcije zasnovane na hešu, jedan od načina za rešavanje ovog problema prelivanja particije je rekurzivna primena tehnike heš spoja na spajanje prepune particije relacije R sa odgovarajućom particijom relacije V. To jest, prvo se dele pariticije relacija R i V na pod-delove, a zatim se pod-delovi udružuju. Svi pod-delovi relacije R verovatno se uklapaju u memoriju. Ako se ipak ne uklapaju, rekurzivno se primenjuje tehnika heš spoja.

### Korišćenje dodatne memorije: Hibridni heš spoj

Minimalna količina memorije potrebna za heš spoj je \* M. Ako je na raspolaganju više memorije, varijanta heš spoja nazvana hibridni heš spoj nudi bolje performanse. Pretpostavka je *B* > *f \** , za neki ceo broj k. To znači da, ako se relacija R podeli na k particija veličine , može se napraviti heš tabela u memoriji za svaku particiju. Za podelu R (slično važi i za relaciju V) na k particija, potrebno je k izlaznih bafera i jedan ulazni bafer, to jest k + 1 stranica. Ovo ostavlja B - (k + 1) dodatnih stranica tokom faze particionisanja.

Pretpostavka je da je *B* - *(k* + 1) > *f*  \* ,to jest, na raspolaganju je dovoljno dodatne memorije tokom faze particioniranja da se u memoriji drži heš tabela za particiju relacije R. Ideja iza hibridnog heš spajanja je izgradnja heš tabele u memoriji za prvu particiju relacije R tokom faze particionisanja, što znači da se ova particija ne zapisuje na disk. Slično tome, dok se particioniše relacija V, umesto se ispisuju torke u prvoj particiji relacije V, može se direktno ispitati tabela u memoriju za prvu R particiju i zapisati rezultati. Na kraju faze particionisanja, završeno je spajanje prvih particija R i V, pored podele dve relacije. U fazi ispitivanja, pridružuju se preostale particije kao u heš spoju.

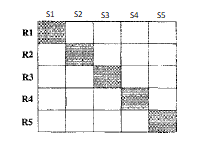
Ušteda ostvarena hibridnim heš spajanjem je ta što se izbegava upisivanje prvih particija R i V na disk tokom faze particionisanja i njihovo ponovno čitanje tokom faze sondiranja. U primeru koji analiziramo, sa 500 stranica u manjoj relaciji R i 1000 stranica u V, ako je na raspolaganju B = 300 stranica, lako se može izgraditi heš tabela u memoriji za prvu particiju R, dok se R deli na dve particije. Tokom faze particionisanja R, skenira se R i ispisuje jedna particiju. Cena je 500 + 250 ako se pretpostavi da su particije jednake veličine. Zatim se skenira V i ispisuje jedna particija. Cena je 1000 + 500. U fazi ispitivanja skenira se druga particija relacija R i V. Cena je 250 + 500. Ukupni troškovi su 750 + 1500 + 750 = 3000. Nasuprot tome, cena heš spoja je 4500.

Ako je na raspolaganju dovoljno memorije za držanje heš tabele u memoriji za sve R, ušteda je još veća. Na primer, ako je *B* > *f* \* *N* + 2, to jest k = 1, može se napraviti heš tabela u memoriji za sve R. To znači da se R čita samo jednom, da bi se igradila ova heš tabela i V čita jednom, za ispitivanje R heš tabele. Cena je 500 + 1000 = 1500.

### Heš spoj ili spoj pomoću blok ugnježdenih petlji

Prilikom predstavljanja algoritma za spajanje pomoću ugnežđenih blok petlji, bilo je reči o ideji izgradnje heš tabele u memoriji za unutrašnju relaciju. Sada se ovaj algoritam spoja pomoću ugnježdenih blok petlji upoređuje sa hibridnim heš spajanjem.

Ako heš tabela za celu manju relaciju stane u memoriju, dva algoritma su identična. Ako su obe relacije velike u odnosu na raspoloživu veličinu bafera, potrebno je nekoliko prolaza preko jedne od relacija u bloku ugnežđenih petlji spoja. Heš spoj je u ovom slučaju efikasnija tehnika. Broj sačuvanih U/I u ovom slučaju korišćenjem algoritma heš spoja u poređenju sa blok ugnežđenim petljama je ilustrovan na slici 14. U drugom slučaju, čitaju se sve torke iz V za svaki blok relacije R. Trošak za U/I odgovara celom pravougaoniku. U algoritmu heš spoja, za svaki blok relacije R, čita se samo odgovarajući blok relacije V. Trošak U/I u tom slučaju odgovara osenčenim površinama na slici. Ova razlika u broju U/I zbog skeniranja relacije V je istaknuta na slici 15.



Slika 14: Heš spoj ili spoj pomoću blok ugnježdenih petlji za velike relacije

Treba napomenuti da je ova slika prilično pojednostavljena. Ne obuhvata troškove skeniranja relacije R u slučaju spoja pomoću blok ugnežđenih petlji i faze particioniranja u heš spoju, i fokusira se na troškove faze ispitivanja.

### Heš spoj ili sortiranje-spajanje spoj

Sada se upoređuje heš spoj sa sotiranje-spajanje algoritmom za spajanje relacija. Ako je na raspolaganju *B* > bafer stranica, gde je M broj stranica u manjoj relaciji i pretpostavi se jednolika veličina particija, cena heš spoja je *3 \* (M* + *N)* U/I. Ako je na raspolaganju *B* > bafer stranica, gde je N broj stranica u većoj relaciji, cena sortiranja-spajanja algoritma spoja je takođe *3 \* (M* + *N*), kao što je objašnjeno u odeljku 5.2.2. Izbor između ovih tehnika stoga zavisi od drugih faktora, naročito:

* Ako particije u heš spoju nisu jednake veličine, heš spoj bi mogao koštati više. Spajanje sortiranjem-spajanjem je manje osetljiv na takvo iskrivljavanje podataka.
* Ako raspoloživi broj bafer stranica padne izmeđui troškovi heš spajanja koštaju manje od sortiranja-spajanja, budući da je potrebno samo dovoljno memorije za držanje particija manje relacije, dok pri spajanju sortiranjem-spajanjem memorijski zahtevi zavise od veličine većeg odnosa. Što je razlika u veličini između dva odnosa veća, ovaj faktor postaje sve važniji.
* Dodatna razmatranja uključuju činjenicu da je rezultat sortiran u sortiranje-spajanje spoju.

# Generalni uslovi spoja

U prethodnom poglavlju je objašnjeno nekoliko algoritama spajanja za slučaj jednostavnog uslova jednakosti za spoj. Drugi važni slučajevi uključuju uslov spajanja koji uključuje jednakosti u nekoliko atributa i uslove nejednakosti. Radi ilustracije slučaja sa nekoliko jednakosti, razmatra se spajanje relacija *Rezervacije* R i *Vozaci* V sa uslovima spoja *R.VID = V.VID ʌ R.RIME = V.VIME*:

* Za spoj pomoću indeks ugnežđenih petlji, može se napraviti indeks na relaciji *Rezervacije*  na kombinaciji polja (*R.VID, R.RIME*) i relacija *Rezervacije* tretirati kao unutrašnja relacija. Takođe može se koristiti postojeći indeks za ovoj kombinaciju polja, ili za *R.VID*, ili za *R.RIME*. Slično važi i za izbor relacije *Vozaci* kao unutrašnje relacije.
* Za spoj sortiranje-spajanje relacija *Rezervacije* se sortira po kombinaciji polja (*VID, RNAME*) i *Vozaci* po kombinaciji polja (*VID, VIME*). Slično, za heš spajanje se dele ove kombinacije polja.
* Ostali algoritmi spajanja o kojima je bilo reči suštinski nisu pogođeni.

Ako je uslov spoja zapravo uslov nejednakosti, na primer, spajanje relacije *Rezervacije* R i *Vozaci* V sa ​​uslovom spajanja *R.RIME* < *V.VIME:*

* Za spoj pomoću indeks ugnežđenih petlji potreban je indeks B+ stabla.
* Heš spoj i sortiranje-spajanje spoj nisu primenljivi.
* Ostali algoritmi spajanja o kojima je bilo reči suštinski nisu pogođeni.

Naravno, bez obzira na algoritam, broj kvalifikovanih torki u spoju nejednakosti verovatno će biti mnogo veći nego u spoju jednakosti.

Izlaganje spajanja se može zaključiti zapažanjem da nijedan algoritam spajanja nije jednolično superiorniji od ostalih. Izbor dobrog algoritma zavisi od veličine relacija koje se spajaju, dostupnih metoda pristupa i veličine bafera. Ovaj izbor može imati značajan uticaj na performanse jer razlika između dobrog i lošeg algoritma za isti spoj može biti ogromna.

# Operacije sa skupovima

U okviru ovog poglavlja se razmatra implementacija skupovnih operacija R ∩ S, R x S, R υ S i R - S. Sa stanovišta implementacije, presek i unakrsni proizvod mogu se posmatrati kao posebni slučajevi spajanja, sa jednakošću na svim poljima kao uslov spoja za presek, i bez uslova spajanja za unakrsni proizvod. Zbog toga nema potrebe dalje obrazlagati ove dve skupovne operacije.

Glavna tačka na koju treba obratiti pažnju u primeni unije je uklanjanje duplikata. Razlika skupova se takođe može primeniti korišćenjem varijacija tehnika za eliminisanje duplikata. Upiti sa unjom i razlikama na jednoj relaciji mogu se smatrati upitom za selekciju sa složenim uslovima selekcije. Tehnike razmotrene u odeljku 3 su primenljive na takve upite.

Postoje dva implementaciona algoritma za uniju i razliku u skupova, ponovo zasnovana na sortiranju i heširanju.

## Pristup zasnovan na sortiranju za uniju i razliku

Za određivanje unije dve relacije R u V primenjuju se sledeći koraci:

* Sortiranje relacije R korišćenjem kombinacije svih polja. Slično, sortira se i relacija V.
* Paralelno skeniranje sortiranih relacija R I V i njihovo spajanje, uz eliminaciju duplikata.

Kao poboljšanje, može se primeniti sortiranje serija relacija R i V i njihovo paralelno spajanje. Ovo usavršavanje je slično onom koje se detaljno razmatra za projekciju. Implementacija R-V je slična. Tokom prolaska za spajanje, zapisujemo samo nizove relacije R u rezultat, nakon provere da se ne pojavljuju u relaciji V.

## Pristup zasnovan na heširanju za uniju i razliku

Da bi se odredila unija dve relacije, R u V, pomoću heširanja primenjuju se sledeći koraci:

1. Podela R i V pomoću heš funkcije h.
2. Obrada svake particije na sledeći način:

* Napravi se heš tabela u memoriji (koristeći heš funkciju h2 *≠*  h) za
* Skenira se . Za svaku torku se istražuje heš tabela za . Ako se torka nalazi u heš tabeli, odbacuje se. U suprotnom, torka se dodaje u tabelu.
* Ispisuje se heš tabela, a zatim se briše da bi se obradila sledeća particija.

Za implementaciju R - V postupa se slično. Razlika je u obradi particije. Nakon izgradnje heš tabele u memoriji za , skenira se . Za svaku torku ispituje se heš tabela. Ako torka nije u tabeli, upisuje se u rezultat.

# Operacije agregacije

SQL upit prikazan na slici 15 uključuje operaciju agregacije za izračunavanje srednje vrednosti, AVG. Druge operacije agregacije podržane u MS SQL Server-u su MIN, MAKS, SUM i COUNT.

15.PNG

Slika 15: Operacija agregacije

Osnovni algoritam za operatore agregacije sastoji se od skeniranja čitave relacije *Vozaci* i održavanja nekih tekućih informacija o skeniranim torkama. Podaci o svakoj operaciji agregacije prikazani su u tabeli 1. Troškovi ove operacije su troškovi skeniranja svih torki relacije *Vozaci*.

|  |  |
| --- | --- |
| **Operacija agregacije** | **Informacije o operaciji** |
| SUM | Suma vraćenih vrednost |
| AVG | Srednja vrednost vraćenih torki |
| COUNT | Ukupan broj vraćenih torki |
| MIN | Najmanja vraćena vrednost |
| MAX | Najveća vraćena vrednost |

Tabela 1: Informacije o funkcijama agregacije

Operatori agregacije se takođe mogu koristiti u kombinaciji sa GROUP BY klauzulom. Ako se upitu na slici 15 doda GROUP BY klauzula na atribut *GODINE\_ISKUSTVA*, morala bi se izračunati prosečna starost vozača za svaku vrednost atributa *GODINE\_ISKUSTVA*. Za upite sa grupisanjem postoje dva dobra algoritma za procenu koji se ne oslanjaju na postojeći indeks. Jedan algoritam je zasnovan na sortiranju, a drugi na heširanju.

Pristup zasnovan na sortiranju je jednostavan. Sortira se relacija prema atributu grupisanja (*GODINE\_ISKUSTVA*), a zatim se ponovo skenira da bi se izračunao rezultat operacije agregacije za svaku grupu. Drugi korak je sličan načinu na koji se implementiraju operacije agregacije bez grupisanja, s tim što je jedina dodatna tačka da se mora paziti na granice grupa. U/I troškovi ovog pristupa su samo troškovi algoritma za sortiranje.

U pristupu heširanja pravi se heš tabela, ako je moguće u glavnoj memoriji, na atributu grupisanja. Dok se relacija skenira, za svaku torku se istražuje heš tabela kako bi se pronašao unos za grupu kojoj ta torka pripada i ažurirale tekuće informacije. Ako se heš tabela uklapa u memoriju, što je verovatno zato što je svaki unos prilično mali i postoji samo jedan unos po vrednosti grupisanja, cena pristupa heširanja je O (M), gde je M veličina relacije.

Ako je relacija toliko velika da se heš tabela ne uklapa u memoriju, relacija se može podeliti pomoću heš funkcije h na vrednost po kojoj se grupiše. S obzirom da se sve torke sa zadatom vrednošću grupisanja nalaze na istoj particiji, svaka particija se može samostalno obraditi izgradnjom heš tabele u memoriji za torke iz nje.

## Implementacija agregacionih operatora korišćenjem indeksa

Tehnika korištenja indeksa za odabir podskupa korisnih torki nije primjenjiva za agregaciju. Međutim, pod određenim uslovima, mogu se efikasno proceniti operacije agregacije korišćenjem indeksa za unos podataka umesto zapisa podataka:

* Ako ključ za pretragu indeksa uključuje sve atribute potrebne za agregatni upit, mogu se primeniti tehnike opisane ranije u ovom odeljku na indeks nad skupom unosa podataka, a ne na prikupljanje zapisa podataka i na taj način izbeći preuzimanje zapisa podataka.
* Ako lista atributa klauzule GROUP BY formira prefiks ključa za pretraživanje indeksa, a indeks je indeks stabla, mogu se dohvatiti unosi podataka (i zapisi podataka, ako je potrebno) redosledom potrebnim za operaciju grupisanja i na taj način izbeći korak sortiranja.

Dati indeks može podržati jednu ili obe ove tehnike.

# Uticaj baferovanja

U implementacijama relacionih operatora, efikasna upotreba bafera je veoma važna, pa se u ovom radu eksplicitno uzima u obzir veličina bafera pri određivanju parametara algoritma za nekoliko razmatranih algoritama. Postoje tri glavne tačke na koje treba obratiti pažnju:

1. Ako se nekoliko operacija izvršava istovremeno, one dele ist bafer. Ovo primetno smanjuje broj bafer stranica dostupnih za svaku operaciju.
2. Ako se torkama pristupa pomoću indeksa, posebno neklasterisanog indeksa, verovatnoća pronalaženja stranice u baferu ako se ona traži više puta zavisi od veličine bafera i politike zamene stranica. Nadalje, ako se torkama pristupa pomoću indeksa koji nije grupisan, svaka dohvaćena torka će verovatno zahtevati da se učita nova stranica. Zbog toga se bafer brzo puni, što dovodi do visokog nivoa aktivnosti straničenja.
3. Ako operacija ima obrazac ponovljenih pristupa stranicama, može se povećati verovatnoća pronalaženja stranice u memoriji dobrim izborom politike zamene ili rezervisanjem dovoljnog broja stranica bafera za operaciju, ukoliko menadžer bafera pruža ovu mogućnost . Sledi nekoliko primera takvih obrazaca ponovljenog pristupa:

* Kod jednostavnog spajanja pomoću ugnežđenih petlji, za svaki par spoljašnje relacije, više puta se skeniraju sve stranice u unutrašnjoj relaciji. Ako ima dovoljno bafer stranica za čuvanje čitave unutrašnje relacije, politika zamene je irelevantna. U suprotnom, politika zamene postaje kritična. Sa LRU[[5]](#footnote-6) politikom zamene stranica nikada se neće pronaći stranica kada se zatraži. Pomoću MRU[[6]](#footnote-7) politike zamene postiže se najbolje iskorištenje bafera - prve B -2 stranice unutrašnje relacije uvek ostaju u baferu.
* U spoju pomoću ugnježdene blok petlje, za svaki blok spoljne relacije skenira se cela unutrašnja relacija. Međutim, pošto je za skeniranje unutrašnje relacije dostupna samo jedna stranica, politika zamene ne uvodi nikakve razlike.
* U algoritmu spoja pomoću indeks ugnežđenih petlji, za svaku torku spoljašnje relacije, koristimo indeks za pronalaženje odgovarajućih unutrašnjih torki. Ako nekoliko torki spoljašnje relacije imaju istu vrednost u atributu spoja, postoji ponovljeni obrazac pristupa na unutrašnjoj relaciji; može se maksimizirati ponavljanje sortiranjem spoljne relacije na atributima spoja.

# Zaključak

Relacioni operatori služe kao gradivni blokovi za obradu upita.

Upit se obično sastoji od nekoliko relacionih operatora. Uopšteno govoreći, fizički operator je implementacija relacionog operatora. Za svaki relacioni operator postoji nekoliko alternativnih algoritama za njegovu implementaciju i ne postoji univerzalno najbolji algoritam implementacije. Zbog toga je mudar odabir fizičkih operatora presudan za performanse upita. Obrađeni su algoritmi koji se koriste za implementaciju operacija selekcije, projekcije, spoja, standardni operatori terije skupova (presek, unija razlika), funkcije agregacije i operator grupisanja.

Zajedničke tehnike koje se koriste za implementaciju relacionih operatora su iteracija, indeksiranje i particionisanje.

Šema relacije baze podataka koja je korišćena u ovom radu sastoji se od tri tabele, *Vozaci*, *Automobili* i *Rezervacije*, a koristi se za rezervaciju automobila u agenciji za iznajmljivanje automobile. Korišćena je Microsoft SQL Server baza podataka.

Implementacija operatora selekcije zavisi od toga da li je relacija sortirana po atributu koji se nalazi u uslovu selekcije, da li postoji indeks na atributu koji se nalazi u uslovu, kao i da li je odgovarajući indeks heš indeks ili indeks B+ stabla.

Za implementaciju operacije projekcije treba ukloniti neželjene atribute i ukloniti sve proizvedene duplicate. Implementacije projekcije može biti zasnovana na sortiranju i heširanju. Microsoft SQL Server baza podataka podržva oba pristupa.

Operacija spoja se implemenira korišćenjem jednostavnih ugnježdenih petlji, blok ugnježdenih petlji, ugnježdenih petlji zasnovanih na indeksu. Takođe, za implementaciju spoja koristi se sortiranje-mešanje algoritam, kao i algoritmi zasnovani na heširanju.

Što se tiče operacija iz teorije skupova, presek i unakrsni proizvod se mogu tretirati kao posebni slučajevi operacija selekcije i spoja. Za implementaciju operacija unije i razlike koriste se pristupi zasnovani na sortiranju i heširanju.

I operacije agregacije se implementiraju korišćenjem pristupa zasnovanog na sortiranju i heširanju. Operacije agregacije se često kombinuju sa GROUP BY klauzulom za grupisanje.

U implementacijama relacionih operatora, efikasna upotreba bafera je veoma važna.

# Literatura

[1] Database Management Systems, 3rd Edition, Raghu Ramakrishnan, Johannes Gehrke, ISBN: 0-07-246563-8

[2] Evaluacija Relacionih Operatora, Prof.dr Leonid Stoimenov, Katedra za računarstvo, EFN, <https://cs.elfak.ni.ac.rs/nastava/pluginfile.php/17936/mod_resource/content/1/Evaluacija%20rel%20operatora%202016.pdf>, datum poslednjeg pristupa 25.9.2021.

[3] Evaluation of Relational Operators, <https://link.springer.com/referenceworkentry/10.1007/978-0-387-39940-9_154>, datum poslednjeg pristupa 25.9.2021

[4] Algoritmi keširanja, <https://sr.wikipedia.org/sr/Algoritmi_ke%C5%A1iranja>, datum poslednjeg pristupa 26.9.2021.

1. Izraz koji koristi logičke veze I (ʌ) i ILI (v) [↑](#footnote-ref-2)
2. Svaki uslov selekcije se može izraziti u CNF-u. [↑](#footnote-ref-3)
3. Kolizija se javlja kada dve torke imaju istu vrednos heš funkcije [↑](#footnote-ref-4)
4. Ovo je tipična cena za indekse zasnovane na hešu [↑](#footnote-ref-5)
5. LRU (engl. *least recently used*) algoritam za zamenu stranica u baferu iz bafera izbacuje stranicu koja najduže nije korišćena [↑](#footnote-ref-6)
6. MRU (engl. Most recently used) algoritam za zamenu stranica koji iz bafera izbacuje stranicu koja je poslednje korišćena [↑](#footnote-ref-7)