Korištenje relacijske algebre u prevođenju upita i izradi upitnoga plana

Galina, Patrik

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Patrik Galina

KORIŠTENJE RELACIJSKE ALGEBRE U PREVOĐENJU UPITA I IZRADI UPITNOG PLANA

ZAVRŠNI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Patrik Galina

Matični broj: 46230/17–R

Studij: Informacijski sustavi

KORIŠTENJE RELACIJSKE ALGEBRE U PREVOĐENJU UPITA I IZRADI UPITNOG PLANA

ZAVRŠNI RAD

Mentor/Mentorica:

Prof. dr. sc. Alen Lovrenčić

Varaždin, lipanj 2020.

Patrik Galina

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog završnog rada je „*Korištenje relacijske algebre u prevođenju upita i izradi upitnog plana*“. Na samom početku reći ćemo nešto o relacijskim bazama podataka koje su danas najrasprostranjenija i najčešće korištena vrsta baza podataka u poslovnom svijetu.
Što su relacijske baze podataka?

Relacijske baze podataka su vrsta baza podataka u kojima su podaci pohranjeni u obliku tablica, odnosno relacija koje su povezane zajedničkim atributima.

U nastavku ćemo se detaljnije baviti relacijskom algebrom te operatorima relacijske algebre.

Što je relacijska algebra?

Robert Manger je u svojoj knjizi *Baze podataka* naveo da se pod pojmom relacijske algebre podrazumijevaju operacije definirane nad relacijama te podacima koji se nalaze unutar samih relacija. Mogli bismo reći da je relacijska algebra srž svakog *sustava za upravljanje bazom podataka (SUBP)*. Temeljni dio relacijske algebre čine relacijski operatori. Pomoću operatora proširene relacijske algebre upit u sintaksi *SQL* deklarativnog jezika reinterpretira se u upit proceduralnog jezika proširene relacijske algebre. Svaki *SQL* upit može se prevesti u jedan ili više povezanih, korespondirajućih upita u relacijskoj algebri.

Što je proširena relacijska algebra?

Proširena relacijska algebra, koju mnogi nazivaju algebrom upita, je redefinirana relacijska algebra koja je uvedena radi usklađivanja definicija *SQL-a* te relacijske algebre. Kod upitnog jezika *SQL*, relacije se klasificiraju kao multiskupovi i liste, dok je klasična relacijska algebra skupovno orijentirana, pa se samim time i relacije klasificiraju kao skupovi slogova. Usklađivanje definicija ovih dvaju jezika nužno je zbog prevođenja upita.

Nakon detaljnog uvida u klasičnu te proširenu relacijsku algebru i njihove operatore, kroz nekoliko primjera bit će objašnjeno prevođenje *SQL* upita u jedan ili više povezanih upita u relacijskoj algebri. U današnje vrijeme se prilikom izvršavanja upita velika pažnja posvećuje vremenskoj komponenti, zbog čega je važno da se upiti optimiziraju kako bi se „skratilo“ vrijeme njihovog izvršavanja. Kroz pravila optimizacije upita bit će prikazana moguća poboljšanja vremena izvršavanja upita.

Ključne riječi: relacijska algebra, proširena relacijska algebra, skupovni operatori, projekcija, selekcija, prirodni spoj, preimenovanje atributa, produkt, aktivni komplement, kvocijent, grupiranje, sortiranje, eliminacija duplikata, prevođenje upita, optimizacija upita

Sadržaj

1. Uvod

U ovom završnom radu osvrnut ćemo se na temu „*Korištenje relacijske algebre u prevođenju upita i izradi upitnog plana*“. Na samom početku reći ćemo nešto o relacijskim bazama podataka te ćemo se nakon toga fokusirati na samu temu ovog završnog rada, relacijsku algebru. Nakon toga ćemo detaljnije objasniti svaki od operatora koji se koriste u relacijskoj algebri, te primjerima objasniti način na koji djeluje pojedini operator. Nakon toga ćemo kroz nekoliko primjera prikazati način pretvorbe *SQL* upita u jedan ili više povezanih upita relacijske algebre te pravila optimizacije upita. Na samom kraju rezimirat ćemo cijelu temu te navesti najvažnije pojmove koje bi trebalo zapamtiti iz ovog područja.

Kako bi mogli shvatiti sam koncept relacijske algebre prvo ćemo ukratko objasniti što su to relacijske baze podataka. „Relacijska baza podataka temelji se na relacijskom modelu baze podataka.“ (*Uvod u baze podataka*, M. Maleković i K. Rabuzin, 2016. , 7. str.)

Model podataka se općenito sastoji od 3 komponente:

1. **strukturalna komponenta** – opisuje način na koji su prikazani podaci
2. **integritetna komponenta** – opisuje ograničenja nad strukturom
3. **operativna komponenta** – opisuje dozvoljene operacije nad strukturom

U relacijskim baza podataka podaci su pohranjeni u obliku tablica, odnosno relacija koje su povezane zajedničkim atributima. U svakoj relaciji možemo imati više ključeva, a jedan od njih odabiremo kao primarni ključ koji jednoznačno definira svaki slog, odnosno redak relacije. „Primarni ključ mora imati svojstvo jedinstvene identifikacije, odnosno svaki redak iz relacije mora imati jedinstvenu vrijednost primarnog ključa.“ (*Uvod u baze* podataka, M. Maleković i K. Rabuzin, 2016. , 7. str.) U relacijama se mogu, ali i ne moraju, pojaviti vanjski ključevi koji se koriste za povezivanje s drugim relacijama. Vanjski ključevi ne moraju imati svojstvo jedinstvene identifikacije, odnosno može se pojaviti više slogova s istom vrijednošću vanjskog ključa. Prilikom povezivanja vanjskog ključa jedne relacije s primarnim ključem druge relacije važno je očuvanje referencijalnog integriteta.

„Referencijalni integritet osigurava logičku vezu i pravila odnosa između podataka u povezanim tablicama.“ (https://www.weboteka.net/fpz/Baze%20podataka/Predavanja/05%20-%20BP%20-%2006.tjedan.pdf, dostupno 23.6.2020.) Definicija referencijalnog integriteta nije intuitivno posve jasna, pa ćemo samu definiciju izraziti jednostavnije. Jednostavnije rečeno, u relaciji ne može postojati vrijednost vanjskog ključa za koju ne postoji vrijednost primarnog ključa u povezanoj relaciji. Iz cijele ove priče vezane za relacijske baze podataka nas zanima samo jedna komponenta relacijskog modela podataka, a to je operativna komponenta koja opisuje dozvoljene operacije nad strukturom.

2. Relacijska algebra

„Relacijsku algebru uveo je Edgar F. Codd u svojim radovima iz 70-ih godina 20. stoljeća.“ (*Baze podataka*, R. Manger , 2014. , 67. str.) Temelji relacijske algebre počivaju na knjizi *A Relational Model of Data for Large Shared Data Banks* čiji autor je upravo Edgar F. Codd.

„Pod pojmom relacijske algebre se podrazumijevaju operacije definirane nad relacijama te podacima koji se nalaze unutar samih relacija.“ (prema *Baze podataka*, R. Manger , 2014. , 68. str.) Operacije koje su definirane nad relacijama te podacima koji se nalaze unutar samih relacija mogu biti unarne, ako djeluju nad jednom relacijom ili binarne, ako djeluju nad dvjema relacijama. Sam koncept relacijske algebre svodi se na računanje vrijednosti različitih algebarskih izraza. Robert Manger u svojoj knjizi *Baze podataka* iz 2014. navodi kako su ti izrazi građeni od unarnih i binarnih operacija, operanada te zagrada. Operandi algebarskih operacija, koji su sastavni dio relacijske algebre, su relacije, a rezultati tih operacija su opet relacije. Kada govorimo o binarnim operacijama tada podrazumijevamo da ta operacija uzima podatke iz nekog skupa te im nakon operacije pridružuje neki podatak koji je iz tog istog skupa. Relacijska algebra se indirektno koristi prilikom postavljanja upita nad bazom podataka. Prilikom postavljanja upita, unutar naredbe *SELECT*, zadana su određena ograničenja nad podacima koji će biti vraćeni kao odgovor na upit. Zadana ograničenja predstavljaju operatore relacijske algebre. U tom pogledu, sama *SELECT* klauzula predstavlja operator projekcije (⊓), *FROM* predstavlja operator produkta, odnosno Kartezijevog produkta (⊗), *WHERE* i *JOIN* predstavljaju operator selekcije (σ), *UNION*, *INTERSECT* i *EXCEPT/MINUS* predstavljaju operatore (∪, ∩, **–**) respektivno.

„Svaki algebarski izraz predstavlja jedan upit u bazu, a njegova vrijednost predstavlja odgovor na upit.“ (*Baze podataka*, R. Manger, 2014. 68. str.). Mirko Maleković i Markus Schatten u svojoj knjizi *Teorija i primjena baza podataka* iz 2017. navode kako se relacijska algebra sastoji od sljedećih temeljnih operatora: presjek, unija, razlika, selekcija, projekcija, preimenovanje atributa, prirodni spoj, Kartezijev produkt, aktivni komplement te kvocijent. Na temelju relacijske algebre nastao je upitni jezik u relacijskim bazama podataka. Relacijska je algebra, za razliku od *SQL-a*, skupovno orijentirana što znači da osigurava jedinstvenost slogova relacije, odnosno pojedini slog može se pojaviti samo jednom u relaciji. Kako bi se riješila neusklađenost definicija uvedena je proširena relacijska algebra o kojoj ćemo detaljnije govoriti u nastavku.

Relacijska algebra primarno izražava značenje upita te plan izvođenja upita u *SUBP-u.*

Sada ćemo na primjeru relacije *studenti* objasniti dijelove od kojih se sastoji svaka relacija.

Tablica 1. Relacija *studenti*

studenti	JMBAG	Prezime	Ime	GodinaStudija
t_1	0151268543	Jerković	Jana	2
t_2	0159315204	Josipović	Josip	3
t_3	0412352152	Filipović	Filip	3
t_4	7250149682	Dorić	Dora	1

Temeljem prikazane relacije *studenti*, naziv relacije je *studenti*. Relacijska shema je prema definiciji konačan, neprazan skup atributa koji predstavlja atribute promatrane relacije. U prikazanom primjeru, relacijska shema se sastoji od atributa *JMBAG*, *Prezime*, *Ime* te *GodinaStudija*. Uređena n-torka atributa se naziva slog ili redak relacije. Relacijsku shemu bi u sintaksi relacijske algebre zapisali kao, *R = {JMBAG, Prezime, Ime, GodinaStudija}*. Svakom od atributa relacijske sheme pridružujemo skup vrijednosti koje atribut može poprimiti, a taj skup vrijednosti nazivamo domenom promatranog atributa. Relacija nad *R* je skup slogova $r = \{t_1, t_2, t_3, t_4\}$, gdje su sa t_1, t_2, t_3, t_4 označeni pojedini redovi (slogovi), odnosno uređene n-torke relacije.

Tablica 2. Relacija *upravljanje*

pilot	PilotId	Ime	Prezime
p_1	1	Ana	Anić
p_2	2	Karlo	Karlić
p_3	3	Marko	Markić

avion	AvionId	Proizvodac	Model
a_1	1	Boeing	787-8
a_2	2	Airbus	A220
a_3	3	Comac	C919

upravljanje	Pilot	Avion
t_1	p_1	a_1
t_2	p_1	a_2
t_3	p_2	a_3
t_4	p_3	a_4

Kroz primjer relacije *upravljanje* objasnit ćemo semantiku relacije, odnosno način na koji se interpretiraju podaci koji se nalaze unutar same relacije. Semantika relacije označava značenje podataka koji se nalaze u relaciji. Semantika sloga ekvivalentna je interpretaciji sloga. (x,y) ∈ *upravljanje* znači da pilot s oznakom x upravlja avionom s oznakom y. Prethodni zapis moguće je zapisati u kraćem obliku, *upravljanje* (x,y), pri čemu semantika zapisa ostaje ista.

Ako semantiku primijenimo na neki od slogova relacije *upravljanje* tada dobivamo sljedeću interpretaciju:

Pilot *Ana Anić* upravlja avionom *Boeing 787-8*.

Možemo primijetiti da je relacija *upravljanje* slabi entitet koji povezuje relacije *pilot* i *avion*, odnosno relacija koja razrješava vezu N:M u podatkovnom modelu.

„Važno je napomenuti kako u relacijama nema ponavljanja slogova, odnosno u relaciji se ne mogu nalaziti dva sloga koja imaju isto semantičko značenje.“ (prema *Teorija i primjena baza podataka*, M. Maleković i M. Schatten, 2017. , 26. str.)

Tablica 3. Relacije *r*(R) i *s*(S)

r	A	C
	1	2
	2	4
	7	2

s	C	A
	2	1
	2	7
	4	2

„Ako su zadane relacije *r*(R) i *s*(S) s atributima A i C, tada su relacije *r* i *s* jednake ako i samo ako vrijedi sljedeće:

1. R ∩ S ≠ ∅, odnosno relacijske sheme nisu međusobno disjunktne te vrijedi da je R = S
2. relacije *r* i *s* se sastoje od istih slogova, pri čemu poredak slogova nije bitan“

(prema *Model podataka i pojam relacije* (prezentacija), 2018. , M. Schatten)

Iz svega navedenog, zaključujemo da je *r* = *s*, odnosno da su relacije *r* i *s* jednake.
Poredak atributa i slogova u relaciji nije bitan, odnosno ako zamijenimo poredak slogova i atributa relacije, kao u prethodnom primjeru, i dalje vrijedi jednakost relacija.

3. Relacijski račun

Kada govorimo o relacijskoj algebri dobro je spomenuti i relacijski račun. Relacijski račun je, kao i relacijsku algebru, uveo Edgar F. Codd u svojim radovima iz 70-ih godina 20. stoljeća. Dakle, možemo primijetiti da je Codd relacijski račun razvijao konkurentno s relacijskom algebrom. Relacijski račun možemo promatrati kao svojevrsnu alternativu relacijskoj algebri.

„Riječ je o matematičkoj notaciji, no ovaj put je ona zasnovana na predikatnom računu“.

(*Baze podataka*, R. Manger, 2014. , 80. str.)

Upit relacijskog računa se temelji na predikatu koji tražene n-torke moraju zadovoljiti kako bi ušle u rezultat.

Razlikuju se dvije vrste relacijskog računa :

1. račun orijentiran na n-torke
2. račun orijentiran na domene

3.1 Račun orijentiran na n-torke

Kada govorimo o relacijskom računu orijentiranom na n-torke tada se pod tim pojmom podrazumijevaju upiti, odnosno algebarski izrazi koji se sastoje od :

- „varijabli n-torki koje poprimaju vrijednosti iz imenovane relacije. Ako je t varijabla koja prolazi relacijom *r*, a D atribut od *r*, tada t.D označava vrijednost od D unutar t
- uvjeta oblika $x \Theta y$, gdje je Θ (*theta*) operator uspoređivanja (=, <, >, ≠, ≤, ≥), pri čemu barem jedan od operanada, *x* ili *y*, mora biti oblika t.D, dok drugi može biti konstanta
- dobro oblikovane formule (*WFF – Well-Formed Function*) koja je građena od logičkih operatora *and*, *or*, *not*, egzistencijalnog kvantifikatora $\exists$ te univerzalnog kvantifikatora $\forall$“

(*Baze podataka*, R. Manger, 2014. , 80. str.)

U nastavku će biti prikazan izraz u sintaksi relacijskog računa orijentiranog na n-torke.

Potrebno je napisati upit koji će pronaći JMBAG-ove, imena te prezimena studenata koji su upisali kolegij „Baze podataka 1“.

{ s.JMBAG, s.Ime, s.Prezime | student(s) and $\exists$ (upisao(u)
and u.JMBAG = s.JMBAG and u.NazivKolegija = 'Baze podataka 1') }

3.2 Račun orijentiran na domene

Razlika u odnosu na račun orijentiran na n-torke očituje se u tome da varijable prolaze domenama, a ne relacijama. „Kod računa orijentiranog na domene javljaju se uvjeti članstva koji imaju oblik *r(A:v_1, B:v_2, C:v_3, ...)*, gdje su A, B, C atributi relacije *r*, a v_1, v_2, v_3 varijable ili konstante.“ (*Baze podataka*, R. Manger, 2014. , 80. str.)
Izrazi koji definiraju pojedini upit i dalje moraju zadovoljavati pravila za *WFF*.

U nastavku će biti prikazan izraz u sintaksi relacijskog računa orijentiranog na domene.
Potrebno je napisati upit koji će pronaći JMBAG-ove, imena te prezimena studenata koji su upisali kolegij „Baze podataka 1“.

{ j, i, p | student(JMBAG : j, ime : i, prezime : p)
and upisao(JMBAG : j, NazivKolegija : 'Baze podataka 1') }

3.3 Odnos relacijskog računa i relacijske algebre

„Relacijski račun i relacijska algebra su jezici koji su ekvivalenti u smislu izražajnosti.“ (*Baze podataka*, R. Manger, 2014. , 80. str.)
Kako su ova dva jezika ekvivalentna, tada se svaki upit zapisan u sintaksi relacijske algebre može pretvoriti u upit u sintaksi relacijskog računa. Zbog ekvivalencije vrijedi i obrat, što znači da se i svaki upit u sintaksi relacijskog računa može pretvoriti u upit u sintaksi relacijske algebre. Važno je napomenuti da ekvivalencija ne ovisi o vrsti relacijskog računa te su pretvorbe moguće bilo iz računa orijentiranog na n-torke ili iz računa orijentiranog na domene. Dokaz ekvivalentnosti relacijske algebre i relacijskog računa nećemo izvoditi, a prve verzije tog dokaza mogu se pronaći u Codd-ovom članku iz 1972. godine.
(https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf, dostupno 28.6.2020.)
Edgar F. Codd je u svojim radovima razvio i algoritam za pretvorbu upita u sintaksi relacijskog računa u upit u sintaksi relacijske algebre koji je poznat pod nazivom redukcijski algoritam.

Relacijski račun možemo smatrati matematičkim model koji predstavlja način na koji bi korisnici trebali postavljati upite, dok relacijska algebra predstavlja način na koji će odabrani *SUBP* interpretirati i izvršavati definirane upite korisnika.

4. Relacijski operatori

„Relacijska algebra sastoji se od relacijskih operatora, koji omogućavaju izračunavanje nove relacije temeljem zadane relacije ili zadanih relacija.“
(*Teorija i primjena baza podataka*, M. Maleković i M. Schatten, 2017. , 26. str.)
Relacijski operatori nam omogućavaju provođenje upita nad bazom podataka jer se na temelju njih izračunava odgovor na postavljeni upit. Za postavljanje upita nad bazom podataka koristi se klauzula *SELECT*, koju smo već i ranije spomenuli.
Pojedine klauzule unutar same naredbe *SELECT*, odnosno klauzule *FROM, WHERE, JOIN, AS, UNION, INTERSECT, EXCEPT/MINUS* koriste se prilikom „izračuna“ odgovora na postavljeni upit nad bazom podataka. Jednostavnije rečeno, prilikom postavljanja upita zadajemo ograničenja podataka koristeći prethodno navedene klauzule, a navedena ograničenja predstavljaju relacijske operatore na temelju kojih „izračunavamo“ odgovor na postavljeni upit. Relacijski operatori ne moraju se koristiti isključivo pojedinačno već se mogu međusobno kombinirati, čime se u suštini iskazuje prava snaga relacijske algebre. To svojstvo relacijskih operatora koristi se prilikom postavljanja složenih upita.
Kombinirajući dosad spomenute operatore relacijske algebre i relacije dobivamo relacijske izraze. Za relacijski izraz kažemo da je regularan ako i samo ako su sve njegove operacije definirane. U suprotnom kažemo da je relacijski izraz neregularan, odnosno da nije dobro definiran. U nastavku ćemo detaljnije objasniti svaki od operatora relacijske algebre.

Primjer 1. Složeni *SQL* upit

```
SQL(u) :    SELECT    A, B
            FROM      r
            UNION
            SELECT    A, B
            FROM      s
```

Navedeni SQL upit bi u terminima relacijske algebre imao sljedeći oblik :

$$\textbf{RA(u)}: \quad \Pi_{A,B}(r) \cup \Pi_{A,B}(s)$$

4.1 Skupovni operatori

Skupovni operatori mogu se koristiti samo u slučaju da relacijske sheme zadanih relacija nisu međusobno disjunktne, odnosno da zadane relacije imaju iste relacijske sheme. Skupovni operatori su binarni operatori, odnosno operatori koji djeluju nad dvjema relacijama. Skupovni operatori relacijske algebre su: *unija*, *presjek*, *razlika*.

4.1.1 Unija

Oznaka operatora: **∪**

„Unija je skupovni relacijski operator koji kao rezultat vraća sve slogove, odnosno n-torke iz jedne relacije i sve n-torke iz druge relacije bez ponavljanja slogova."
(prema *Baze podataka*, R. Manger, 2014. , 70. str.)
Važno je napomenuti da unija ne mijenja relacijsku shemu relacije. Možemo reći da je unija dviju relacija skup svih slogova koji se nalaze ili u jednoj relaciji ili u drugoj relaciji ili u obje relacije. Unija je komutativni operator što znači da je rezultat operacije neovisan o redoslijedu relacija koje sudjeluju u operaciji, odnosno $r \cup s = s \cup r$, gdje su r i s relacije.

Primjer 2. Skupovni operator *unija* ($r \cup s$)

r	A	B	C
	3	4	6
	5	5	9
	1	7	2

s	A	B	C
	1	2	9
	3	4	6
	1	7	2

r ∪ s – slogovima iz relacije *r* dodajemo slogove iz relacije *s*, ali bez ponavljanja slogova. Odnosno ako se u obje relacije nalazi isti slog, tada ćemo taj slog dodati samo jednom dok će „duplikat" sloga biti izbačen iz rezultata.

r ∪ s	A	B	C
	3	4	6
	5	5	9
	1	7	2
	1	2	9

4.1.2 Presjek

Oznaka operatora: ∩

Presjek je skupovni relacijski operator koji vraća sve slogove koji su zajednički u relacijama, odnosno sve n-torke iz jedne relacije koje se pojavljuju i u drugoj relaciji bez ponavljanja slogova. Presjek ne mijenja relacijsku shemu relacije. Presjek je komutativni operator što znači da je rezultat operacije neovisan o redoslijedu relacija koje sudjeluju u operaciji, odnosno $r \cap s = s \cap r$, gdje su *r* i *s* relacije.

Primjer 3. Skupovni operator *presjek* ($r \cap s$)

r	A	B	C
	4	1	2
	2	1	9
	8	1	3
	4	2	2

s	A	B	C
	4	2	2
	1	7	4
	8	1	3
	2	1	9
	5	9	1

r ∩ s – presjek relacija *r* i *s* je skup koji se sastoji od slogova koji su zajednički jednoj i drugoj relaciji. Duplikati slogova se eliminiraju iz rezultantne relacije. Rezultat presjeka relacija *r* i *s* je skup svih slogova iz relacije *r* koji se nalaze u relaciji *s* ili obrnuto.

r ∩ s	A	B	C
	2	1	9
	8	1	3
	4	2	2

4.1.3 Razlika

Oznaka operatora: **–**

„Razlika je skupovni operator koji kao rezultat vraća sve slogove iz prve relacije uz eliminaciju slogova koji su zajednički u obje relacije.“
(prema *Baze podataka*, R. Manger, 2014. , 70. str.)
Razlika ne mijenja relacijsku shemu relacije. Razlika je skup svih slogova koji se nalaze u jednoj relaciji, a nisu i u drugoj relaciji. Kod razlike se prilikom ponavljanja slogova oba sloga odbacuju te ne ulaze u rezultat. Kardinalnost (broj pojava) nekog sloga u rezultantnoj relaciji jednaka je razlici kardinalnosti tog sloga u svakoj od relacija. Ukoliko definiciju kardinalnosti primijenimo na operator razlike u klasičnoj relacijskoj algebri, to znači da će kardinalnost svakog sloga biti 0 ili 1. Za razliku od unije i presjeka, razlika nije komutativni operator što znači da je rezultat operacije ovisan o redoslijedu relacija koje sudjeluju u operaciji.

Primjer 4. Skupovni operator razlika (*r* – *s*)

r	A	B	C
	3	1	7
	1	9	4
	2	2	8
	8	4	0

s	A	B	C
	3	7	5
	2	2	8
	6	4	1
	2	8	9

r – s – razlika relacija *r* i *s* je skup koji se sastoji od slogova iz prve relacije uz eliminaciju slogova koji su zajednički, odnosno slogova koji se pojavljuju u obje relacije. U terminologiji relacijske algebre mogli bismo reći da je razlika skup svih slogova iz prve relacije uz eliminaciju presjeka relacija koje sudjeluju u operaciji.

r – s	A	B	C
	3	1	7
	1	9	4
	8	4	0

4.2 Projekcija

Oznaka operatora: Π

Do sada smo govorili o binarnim operatorima (*unija, presjek, razlika*), odnosno operatorima koji se primjenjuju nad dvjema relacijama. Za razliku od skupovnih operatora, projekcija je unarni operator što znači da se primjenjuje na jednoj relaciji.

„Projekcija je unarni operator koji iz relacije izvlači zadane atribute, s time da se u rezultirajućoj relaciji eliminiraju n-torke duplikati.“ (*Baze podataka*, R. Manger, 2014. , 72. str.)

Projekcija ima veći prioritet od ostalih relacijskih operatora, osim ako zagradama nije drugačije označeno. Projekciju relacije *s*(R), čija je relacijska shema *R = {A, B, C}* označavamo sa Π_R, odnosno $\Pi_{A, B, C}$. Projekcija mijenja relacijsku shemu relacije. Općenito, projekciju možemo promatrati kao vertikalni filter jer projekcijom mijenjamo broj atributa, odnosno stupaca rezultantne relacije.

Primjer 5. Projekcija ($\Pi_{A,C,D}(r)$)

r	A	B	C	D
	2	8	5	4
	1	9	5	6
	4	4	7	2
	7	4	2	8
	2	6	5	4

$\Pi_{A,C,D}$(r) – projekcija relacije *r* je skup koji se sastoji od svih slogova iz relacije *r*, bez ponavljanja, uz promjenu relacijske sheme tog skupa. Relacijska shema rezultantne relacije sastoji se od atributa koji su definirani projekcijom.

$\Pi_{A,C,D}$(r)	A	C	D
	2	5	4
	1	5	6
	4	7	2
	7	2	8

4.3 Selekcija

Oznaka operatora: σ

„Selekcija je unarni operator koji izvlači iz relacije one n-torke koje zadovoljavaju zadani Booleov uvjet. Selekcija ima definiran Booleov uvjet koji se navodi kao *subscript* operatora." (prema http://grdelin.phy.hr/~ivo/Nastava/Baze_podataka/predavanja-2004/06_pred.pdf, dostupno 23.6.2020.)

Booleov uvjet je formula koja se sastoji od :

- konstanti ili atributa
- operatora za uspoređivanje
- logičkih operatora (AND, OR, NOT)

„U suštini, Booleov uvjet je kombinacija izraza koji su povezani logičkim operatorima *and* (∧) ili *or* (∨). Izrazi Booleovog uvjeta imaju format :

atribut operator konstanta |
atribut operator atribut,

pri čemu se *atribut* referencira prema poziciji (.*j* ili *j*) ili prema nazivu (.*naziv* ili *naziv*), a *operator* nekim od operatora uspoređivanja (<, ≤, =, >, ≥, ≠)."
(*Database Management System: Third Edition*, R. Ramakrishnan i J. Gehrke, 2003. , 103. str.)

Selekcija ne mijenja relacijsku shemu relacije. Selekcija je skup svih slogova iz relacije koji zadovoljavaju postavljeni Booleov uvjet. U selekciji nema ponavljanja slogova. Općenito, selekciju možemo promatrati kao horizontalni filter, jer selekcijom mijenjamo broj slogova (redova) rezultantne relacije.

Selekcija ima vrlo značajnu ulogu u proširenoj relacijskoj algebri. Jedan od temeljnih operatora proširene relacijske algebre, Θ spoj, je implementiran korištenjem Kartezijevog produkta te selekcije.

Primjer 6. Selekcija ($\sigma_F(r)$)

r	A	B	C	D
t_1	1	a	2	b
t_2	2	b	1	a
t_3	9	a	7	c
t_4	7	c	4	a
t_5	8	a	5	c

Uvjet koji pojedini slog mora zadovoljiti definiran je s *F.*

$$F: (A > C) \wedge (D \neq a)$$

σ_F(r) – selekcija relacije *r* prema zadanom Boolean uvjetu *F* je skup koji se sastoji od svih slogova iz relacije *r* koji zadovoljavaju uvjet *F*.

σ_F(r)	A	B	C	D
t_3	9	a	7	c
t_5	8	a	5	c

t_1: $(1 > 2) \wedge (b \neq a) \equiv 0 \wedge 1 \equiv 0$

t_2: $(2 > 1) \wedge (a \neq a) \equiv 1 \wedge 0 \equiv 0$

t_3: $(9 > 7) \wedge (c \neq a) \equiv 1 \wedge 1 \equiv 1$

t_4: $(7 > 4) \wedge (a \neq a) \equiv 1 \wedge 0 \equiv 0$

t_5: $(8 > 5) \wedge (c \neq a) \equiv 1 \wedge 1 \equiv 1$

4.4 Prirodno spajanje

Oznaka operatora: ⋈

Prirodni spoj je binarna operacija koja je primjenjiva na bilo koje dvije relacije neovisno o relacijskim shemama.

Operator prirodnog spoja smatra se temeljnim operatorom klasične relacijske algebre iako u praksi nema značajnu primjenu. U osnovi, prirodni spoj se temelji na povezivanju relacija po jednakosti istoimenih atributa. Prilikom dizajna baze podataka vrlo se rijetko pojedini atributi relacija koje sudjeluju u spajanju jednako zovu zbog čega se prirodno spajanje u praksi zanemaruje te zamjenjuje *theta* spojem.

Neka su zadane relacije *r* i *s*. Prirodni spoj kao rezultat vraća sve kombinacije slogova iz prve relacije sa svim slogovima iz druge relacije pri čemu razlikujemo dva slučaja :

- ako su relacije *r* i *s* međusobno disjunktne, odnosno ako relacijske sheme relacija *r* i *s* nemaju zajedničkih atributa tada prirodni spoj vraća sve kombinacije slogova iz relacije *r* sa svim slogovima iz relacije *s*. Relacijska shema rezultantne relacija je unija relacijskih shema relacija *r* i *s* što u terminologiji relacijske algebre možemo zapisati kao $R(r \bowtie s) = R(r) \cup R(s)$. Slogovi rezultantne relacije su Kartezijev produkt slogova relacija *r* i *s*.
- ako relacije *r* i *s* nisu međusobno disjunktne, odnosno ako relacijske sheme relacija *r* i *s* imaju zajedničkih atributa tada prirodni spoj povezuje samo one slogove iz prve relacije sa slogovima iz druge relacije koje imaju iste vrijednosti zajedničkih atributa. Relacijska shema rezultantne relacija je unija relacijskih shema relacija *r* i *s* pri čemu se zajednički atributi pojavljuju samo jednom.

(prema *Relacijski operatori* (prezentacija), 2018. , M. Schatten)

„U rezultirajućoj relaciji zajednički atribut pojavljuje se samo jednom što znači da se duplikati atributa odbacuju iz rezultantne relacije.“ (*Baze podataka*, R. Manger, 2014. , 76. str.)

Primjer 7. Prirodno spajanje ($r \bowtie s$) – relacije s međusobno disjunktnim relacijskim shemama

r	A	B	C
	2	1	9
	1	7	6
	9	2	5

s	D	E
	6	1
	1	4

r ⋈ s – prirodni spoj relacija *r* i *s*, s međusobno disjunktnim shemama, je skup koji se sastoji od svih kombinacija slogova iz relacije *r* sa svim slogovima iz relacije *s*.

r ⋈ s	A	B	C	D	E
	2	1	9	6	1
	2	1	9	1	4
	1	7	6	6	1
	1	7	6	1	4
	9	2	5	6	1
	9	2	5	1	4

Primjer 8. Prirodno spajanje ($r \bowtie s$) – relacije s međusobno nedisjunktnim relacijskim shemama

r	A	B	C
	1	a	b
	7	b	a
	5	c	a

s	B	C	D
	a	b	2
	a	b	9
	c	a	7

r ⋈ s – prirodni spoj relacija *r* i *s* čije relacijske sheme **nisu** međusobno disjunktne, odnosno relacija koje imaju zajedničkih atributa, je skup koji se sastoji samo od kombinacija slogova iz prve relacije sa slogovima iz druge relacije koje imaju iste vrijednosti zajedničkih atributa.

r ⋈ s	A	B	C	D
	1	a	b	2
	1	a	b	9
	5	c	a	7

4.5 Preimenovanje atributa

Oznaka operatora: δ

Preimenovanje atributa je operacija koja mijenja imena atributa u relacijskoj shemi relacije. Postoji nekoliko osnovnih razloga za promjenu naziva atributa u relacijskoj shemi relacije. Prilikom izrade baze podataka može se ustanoviti da za neke atribute nismo izabrali odgovarajuća imena pa je iz tog razloga potrebno odraditi preimenovanje atributa.
„Budući da neki atribut može biti u više različitih relacijskih shema, onda njegovo referenciranje može zahtijevati preciziranje relacijske sheme u kojoj se dani atribut nalazi.“
(*Teorija i primjena baza podataka*, M. Maleković i M. Schatten, 2017. , 31. str.)
Prethodno navedeni razlog ćemo sada objasniti na jednostavnom primjeru. Neka su zadane relacije *r* i *s* s relacijskim shemama *R = {A, B, C}* i *S = {B, C, D}*. Relacije *r* i *s* se podudaraju na atributima B i C pa se prilikom njihovog referenciranja specificira relacija u kojoj se taj atribut nalazi, odnosno atributi B i C se preimenuju u *r*.B i *r*.C te *s*.B i *s*.C.

Primjer 9. Preimenovanje atributa ($\delta_{D\leftarrow F}(r)$)

r	A	B	C	D
	2	a	8	9
	1	c	5	2
	9	b	1	5
	5	c	3	7

$\delta_{D\leftarrow F}$ – preimenovanje atributa mijenja relacijsku shemu relacije tako da mijenja naziv atributa koji se nalazi s lijeve strane izraza s nazivom atributa koji se nalazi s desne strane izraza.

$\delta_{D\leftarrow F}$	A	B	C	F
	2	a	8	9
	1	c	5	2
	9	b	1	5
	5	c	3	7

4.6 Produkt (Kartezijev produkt)

Oznaka operatora: $\otimes$

> „Neka su zadane relacije *r*(R) i *s*(S). Produkt relacija *r* i *s* definiramo kao :
>
> 1. Ako je $R \cap S = \emptyset$, onda je $r \otimes s = r \bowtie s$.
> 2. Ako je $R \cap S \neq \emptyset$, onda izvršimo preimenovanje atributa u R i S tako da postignemo disjunktnost, a zatim primijenimo 1. korak“

(*Teorija i primjena baza podataka*, M. Maleković i M. Schatten, 2017. , 35. str.)

Kartezijev produkt je jedan od najčešće korištenih relacijskih operatora u bazama podataka. Ukoliko su relacijske sheme relacija međusobno disjunktne tada je Kartezijev produkt jednak relacijskoj operaciji prirodnog spoja. Kartezijev produkt se obično implementira uz neki uvjet selekcije. Na taj način je ostvareno da se slogovi rezultata, dobiveni operacijom Kartezijevog produkta, filtriraju prema određenom kriteriju.

Kartezijev produkt je komutativni operator što znači da je rezultat operacije neovisan o redoslijedu relacija koje sudjeluju u operaciji, odnosno $r \otimes s = s \otimes r$, gdje su *r* i *s* relacije.
Kartezijev produkt ima vrlo značajnu ulogu u proširenoj relacijskoj algebri. Jedan od temeljnih operatora proširene relacijske algebre, Θ spoj, je implementiran korištenjem Kartezijevog produkta te selekcije.

Iako ima vrlo značajnu ulogu u proširenoj relacijskoj algebri, Kartezijev produkt se izbjegava te zamjenjuje drugim operatorima kad god i koliko god je to moguće. Razlog tome je složenost operacije Kartezijevog produkta. Naime, zbog kvadratne prostorne složenosti, količina potrebne memorije vrlo brzo raste što može uzrokovati značajnu degradaciju performansi prilikom izvođenja upita.

Primjer 10. Kartezijev produkt (*r* ⊗ *s*)

r	A	B	C	D
	6	a	b	4
	3	c	b	1
	9	b	a	2
	5	c	c	7
	2	a	c	6

s	B	C	E
	a	b	2
	a	b	9
	c	a	7

r ⊗ s – Kartezijev produkt relacija *r* i *s* je skup koji se sastoji od svih kombinacija slogova iz prve relacije sa slogovima iz druge relacije pri čemu je potrebno preimenovati zajedničke atribute ukoliko relacijske sheme nisu međusobno disjunktne.

r ⊗ s	A	r.B	r.C	D	s.B	s.C	E
	6	a	b	4	a	b	2
	6	a	b	4	a	b	9
	6	a	b	4	c	a	7
	3	c	b	1	a	b	2
	3	c	b	1	a	b	9
	3	c	b	1	c	a	7
	9	b	a	2	a	b	2
	9	b	a	2	a	b	9
	9	b	a	2	c	a	7
	5	c	c	7	a	b	2
	5	c	c	7	a	b	9
	5	c	c	7	c	a	7
	2	a	c	6	a	b	2
	2	a	c	6	a	b	9
	2	a	c	6	c	a	7

4.7 Aktivni komplement

Oznaka operatora: AC

„Aktivni komplement je unarni operator, koji omogućava dobivanje, u određenom smislu, negativne informacije. Radi se o tome da možemo biti zainteresirani za neku informaciju koja nije eksplicite navedena u danoj relaciji.“
(*Teorija i primjena baza podataka*, M. Maleković i M. Schatten, 2017. , 39. str.)

Aktivni komplement kao rezultat vraća skup koji se sastoji od kombinacija slogova koji nisu navedeni u polaznom skupu. Sam relacijski operator je sličan negaciji razlike jer kao rezultat vraća sve permutacije koje nisu zadane u početnoj relaciji.

Neka je zadana relacija *r* s relacijskom shemom *R = {A, B, C, D}*. Aktivni komplement relacije *r* možemo zapisati na sljedeći način :

$$AC(r) = (\Pi_A(r) \bowtie \Pi_B(r) \bowtie \Pi_C(r) \bowtie \Pi_D(r)) - r$$

Primjer 11. Aktivni komplement (AC(*r*))

r	A	B	C	D
	4	2	a	b
	1	2	b	c
	9	2	a	a

AC(r) – aktivni komplement relacije *r* je skup koji se sastoji od svih kombinacija slogova koji se ne nalaze u polaznoj relaciji. Prvo je potrebno napraviti prirodni spoj između projekcija atributa iz polazne relacije. Kako bi dobili aktivni komplement, iz skupa uklonimo sve slogove koji se nalaze u polaznoj relaciji. U terminologiji relacijske algebre to možemo zapisati u obliku $AC(r) = (\Pi_A \bowtie \Pi_B \bowtie \Pi_C \bowtie \Pi_D) - r$.

Postupak kojim izračunavamo aktivni komplement može se podijeliti u nekoliko koraka :

1. projekcija atributa iz relacije *r*
2. prirodni spoj projekcija atributa iz prethodnog koraka
3. razlika relacije dobivene u prethodnom koraku i polazne relacije

Kod izračuna aktivnog komplementa vodit ćemo se navedenim koracima kako bi postupak bio što jednostavniji i razumljiviji.

Koraci izračuna aktivnog komplementa :

1. Projekcija atributa iz relacije *r*

$\Pi_A(r)$	A
	4
	1
	9

$\Pi_B(r)$	B
	2

$\Pi_C(r)$	C
	a
	b

$\Pi_D(r)$	D
	b
	c
	a

Kako je aktivni komplement operator klasične relacijske algebre koja je skupovno orijentirana, duplikati slogova se „odbacuju“ iz rezultantne relacije. U navedenom primjeru prilikom projekcije atributa B, slog s vrijednošću *2* pojavio se više puta te je zbog toga duplikat sloga odbačen iz rezultata.
Isto vrijedi i za projekciju atributa C kod koje se slog s vrijednošću '*a*' pojavio dva puta te je duplikat sloga odbačen iz rezultata.

(r)

2. Prirodni spoj projekcija atributa iz prethodnog koraka

$\Pi_A(r) \bowtie \Pi_B(r) \bowtie \Pi_C(r) \bowtie \Pi_D(r)$	A	B	C	D
	4	2	a	b
	4	2	a	c
	4	2	a	a
	4	2	b	b
	4	2	b	c
	4	2	b	a
	1	2	a	b
	1	2	a	c
	1	2	a	a
	1	2	b	b
	1	2	b	c
	1	2	b	a
	9	2	a	b
	9	2	a	c
	9	2	a	a
	9	2	b	b
	9	2	b	c
	9	2	b	a

3. Razlika relacije dobivene u prethodnom koraku i polazne relacije

AC(r)	A	B	C	D
	4	2	a	c
	4	2	a	a
	4	2	b	b
	4	2	b	c
	4	2	b	a
	1	2	a	b
	1	2	a	c
	1	2	a	a
	1	2	b	b
	1	2	b	a
	9	2	a	b
	9	2	a	c
	9	2	b	b
	9	2	b	c
	9	2	b	a

Kako bi aktivni komplement bio lakše savladan, u nastavku ćemo prikazati postupak izračuna na konkretnom primjeru.

Primjer 12. Aktivni komplement(AC(*r*))

posjetio	Osoba	Drzava
	Ivo	Rusija
	Ana	Njemačka
	Ana	Austrija
	Ivo	Njemačka

Potrebno je odrediti koje od država **nije** posjetio Ivo te koje od država **nije** posjetila Ana. Dakle, trebamo odrediti sve kombinacije slogova iz relacije *posjetio* koji nisu sadržani u polaznoj relaciji, a upravo to nam omogućava operacija aktivnog komplementa.

Koraci izračuna aktivnog komplementa :

1. Projekcija atributa iz relacije *posjetio*

Π_{Osoba}(posjetio)	Osoba
	Ivo
	Ana

Π_{Drzava}(posjetio)	Drzava
	Rusija
	Njemačka
	Austrija

2. Prirodni spoj projekcija atributa iz prethodnog koraka

Π_{Osoba}(posjetio) ⋈ Π_{Drzava}(posjetio)	Osoba	Drzava
	Ivo	Rusija
	Ivo	Njemačka
	Ivo	Austrija
	Ana	Rusija
	Ana	Njemačka
	Ana	Austrija

3. Razlika relacije dobivene u prethodnom koraku i polazne relacije

s_1	Osoba	Drzava
	Ivo	Rusija
	Ivo	Njemačka
	Ivo	Austrija
	Ana	Rusija
	Ana	Njemačka
	Ana	Austrija

−

posjetio	Osoba	Drzava
	Ivo	Rusija
	Ana	Njemačka
	Ana	Austrija
	Ivo	Njemačka

$s_1 = \Pi_{Osoba}(posjetio) \bowtie \Pi_{Drzava}(posjetio)$

s_1 – posjetio – razlika relacija *s_1* i *posjetio* je skup koji se sastoji od slogova iz prve relacije uz eliminaciju slogova koji su zajednički, odnosno slogova koji se pojavljuju u obje relacije. U terminologiji relacijske algebre mogli bismo reći da je razlika skup svih slogova iz prve relacije uz eliminaciju presjeka relacija koje sudjeluju u operaciji.

AC(posjetio)	Osoba	Drzava
	Ivo	Austrija
	Ana	Rusija

Kako se u polaznoj relaciji *posjetio* nalaze slogovi čija interpretacija ukazuje na države koje je pojedina osoba posjetila, tada aktivni komplement pokazuje upravo suprotno, odnosno ukazuje na države koja pojedina osoba **nije** posjetila.
Interpretacija slogova dobivenog aktivnog komplementa je sljedeća :

Ivo **nije** posjetio Austriju, a Ana **nije** posjetila Rusiju.

4.8 Kvocijent

Oznaka operatora: ÷

„Kvocijent relacija *r*(R) i *s*(S) predstavlja skup svih slogova *t* iz ⊓T(*r*) takvih da kad spojimo slog iz *t* s bilo kojim slogom t_1 iz *s*(S) dobijemo slog iz *r*, gdje je *T = R – S*.“
(*Teorija i primjena baza podataka,* M. Maleković i M. Schatten, 2017. , 44. str.)
Kvocijent relacija *r* i *s* je definiran samo u slučaju da je S pravi podskup od R, odnosno samo ako S sadrži jedan element manje od R. Kvocijent je složeni relacijski operator. Kvocijent služi za provjeru da li se neki skup nalazi unutar nekog drugog skupa.

Primjer 13. Kvocijent (*r ÷ s*)

r	#student	NazivKolegija
	1	Baze podataka 2
	1	Programiranje 1
	2	Baze podataka 2
	2	Operacijski sustavi 1

Semantika relacije *r* je sljedeća. Student sa šifrom *#student* je upisan na kolegij *NazivKolegija*. Ako trebamo odrediti sve studente koji su upisali kolegije *Baze podataka 2* i *Operacijske sustave 1* tada relacijski operator kvocijenta ima glavnu uloga. Zadani kolegiji, *Baze podataka 2* i *Operacijske sustave 1*, predstavljaju ograničenja koja slogovi u rezultantnoj relaciji moraju zadovoljiti. Prvi korak je uvođenje dodatne, pomoćne relacije *s* čiji slogovi su prethodno definirani „uvjeti“.

s	NazivKolegija
	Baze podataka 2
	Operacijski sustavi 1

Studente koji su upisali kolegije *Baze podataka 2* i *Operacijske sustave 1* dobivamo tako da izračunamo kvocijent *r ÷ s.*

r ÷ s	#student
	2

5. Proširena relacijska algebra

Proširena relacijska algebra, koju mnogi nazivaju algebrom upita, je redefinirana relacijska algebra koja je uvedena radi usklađivanja definicija *SQL-a* te relacijske algebre. Algebra upita koristi se prilikom izrade upitnih planova u bazama podataka.

Kod upitnog jezika *SQL*, relacije se klasificiraju kao multiskupovi i liste, dok je klasična relacijska algebra skupovno orijentirana, pa se samim time i relacije klasificiraju kao skupovi slogova. Proširenja relacijske algebre su potrebna kako bi se relacijska algebra uskladila s upitnom (*Query Language*) komponentom *SQL* jezika.

Sintaksa relacijske algebre je jednostavnija i čišća od sintakse *SQL-a* zbog čega je lakša za daljnju obradu. Osim toga, operatori relacijske algebre se bolje podudaraju s algoritmima koji se kasnije koriste prilikom izvršavanja upita pa je pomoću relacijske algebre puno jednostavnije izraditi logički i fizički plan upita te interpretirati upit.

Klasična relacijska algebra ne može se izravno koristiti kod prevođenja *SQL* upita u upit relacijske algebre iz više razloga. Jedan od razloga je što klasična relacijska algebra definira samo osnovni, odnosno temeljni skup operatora koji nisu dostatni za definiranje cijelog skupa *SQL* klauzula. Drugi razlog je skupovna orijentacija klasične relacijske algebre. *SQL* je deklarativni upitni jezik orijentiran na multiskupove i liste zbog čega dolazi do neusklađenosti definicija tih dvaju jezika.

Liste su vrlo važne jer se jedino pomoću njih može pravilo teorijski, a sukladno tome i praktično interpretirati sortiranje u upitima.

Zbog svega navedenog relacijsku algebru je potrebno redefinirati kako bi bila definirana na multiskupovima i listama te na taj način zadovoljila definiciju *SQL* upitnog jezika.

Kako ekspresivna snaga relacijske algebre također odstupa od *SQL-a* potrebno je definirati dodatne operatore. Operatori koji se uvode u proširenoj relacijskoj algebri su operator sortiranja, operator grupiranja te operator eliminacije duplikata.

Osim definiranja novih operatora, javlja se potrebna za izbacivanjem operatora koji nemaju praktičnu važnost te ne interpretiraju klauzule *SQL* upitnog jezika. Operatori koji se izbacuju iz proširene relacijske algebre su operatori preimenovanja, prirodnog spoja, kvocijenta te aktivnog komplementa.

Proširena relacijska algebra kao takva nije zadovoljavala sve zahtjeve *SQL* upitnog jezika zbog čega su dodani i novi operatori uspoređivanja (*LIKE, BETWEEN, ANY, ALL, IN*). Na taj način je definicija relacijska algebra u potpunosti usklađena s definicijom *SQL* upitnog jezika.

6. Operatori proširene relacijske algebre

U prethodnom poglavlju je rečeno kako su proširenja relacijska algebre potrebna kako bi se klasična relacijska algebra uskladila s upitnom (*Query Language*) komponentom *SQL* jezika. Kako se definicija klasične relacijske algebre temelji na skupovima, tada su i njezini operatori definirani na skupovima te ih je u svrhu usklađivanja s *SQL-om* potrebno redefinirati. Rekli smo da ekspresivna snaga relacijske algebre također odstupa od *SQL-a*, pa je potrebno definirati dodatne operatore.

Operatori koji se uvode u proširenoj relacijskoj algebri su operator sortiranja, operator grupiranja te operator eliminacije duplikata.

Osim definiranja novih operatora, javlja se potrebna za izbacivanjem operatora koji nemaju praktičnu važnost te ne interpretiraju klauzule *SQL* upitnog jezika.

Operatori koji se izbacuju iz proširene relacijske algebre su operatori preimenovanja, prirodnog spajanja, kvocijenta te aktivnog komplementa. Operator projekcije u proširenoj relacijskoj algebri implicitno definira preimenovanje atributa, zbog čega se operator preimenovanja izbacuje u proširenoj relacijskoj algebri jer uzrokuje redundanciju. Operator prirodnog spoja smatra se temeljnim operatorom klasične relacijske algebre iako u praksi nema značajnu primjenu. U osnovi, prirodni spoj se temelji na povezivanju relacija po jednakosti istoimenih atributa. Prilikom dizajna baze podataka vrlo se rijetko pojedini atributi relacija koje sudjeluju u spajanju jednako zovu zbog čega se prirodno spajanje u proširenoj relacijskoj algebri izbacuje te zamjenjuje *theta* spojem. Operatori kvocijenta i aktivnog komplementa se izbacuju iz razloga jer se koriste samo u specijalnim situacijama te se puno jednostavnije i efikasnije definiraju putem drugih operatora relacijske algebre. Primjerice, aktivni komplement je značajno efikasniji ukoliko se definira preko operatora razlike.

Izbacivanjem operatora preimenovanja oslobađa se simbol δ koji u proširenoj relacijskoj algebri definira operator eliminacije duplikata.

Dakle, osnovni skup operatora se proširuje novim operatorima čime se povećava i skup klauzula naredbe *SELECT* koje mogu biti interpretirane u relacijskoj algebri.

U nastavku ćemo redefinirati pojedine operatore klasične relacijske algebre.

6.1 Skupovni operatori

Skupovni operatori proširene relacijske algebre se definiraju i interpretiraju na ponešto drugačiji način nego u klasičnoj relacijskoj algebri. Skupovni operatori se i dalje mogu koristiti samo u slučaju da relacijske sheme zadanih relacija nisu međusobno disjunktne, odnosno ako zadane relacije imaju iste relacijske sheme. Kako se definicija proširene relacijska algebre temelji na multiskupovima, tada su i skupovni operatori definirani na multiskupovima. Koncept skupovnih operatora je u osnovi isti uz razliku da kardinalnost pojedinog elementa multiskupa može biti veća od jedan.

6.1.1 Unija

Oznaka operatora: **∪**

Definicija unije ostaje ista kao i kod klasične relacijske algebre, uz razliku da su relacije definirane kao multiskupovi zbog čega nema eliminacije duplikata slogova. Kardinalnost pojedinog sloga jednaka je zbroju kardinalnosti[1] sloga u svakoj od relacija.

Primjer 14. Skupovni operator *unija* (*r* ∪ *s*)

r	A	B	C
	3	4	6
	5	5	9
	1	7	2

s	A	B	C
	1	2	9
	3	4	6

r ∪ s – slogovima iz relacije *r* dodajemo slogove iz relacije *s* pri čemu se pojedini slog može pojaviti više puta.

r ∪ s	A	B	C
	3	4	6
	5	5	9
	1	7	2
	1	2	9
	3	4	6

[1] Kardinalnost sloga označava broj pojava sloga u multiskupu.

6.1.2 Presjek

Oznaka operatora: ∩

Definicija presjeka ostaje ista kao i kod klasične relacijske algebre, uz razliku da su relacije definirane kao multiskupovi zbog čega nema eliminacije duplikata slogova. Kardinalnost pojedinog sloga jednaka je minimalnoj kardinalnosti sloga u nekoj od relacija.

Primjer 15. Skupovni operator *presjek* (*r* ∩ *s*)

r	A	B	C
	4	1	2
	2	1	9
	8	1	3

s	A	B	C
	2	1	9
	1	7	4
	8	1	3
	2	1	9

r ∩ s – presjek relacija *r* i *s* je multiskup koji se sastoji od slogova koji su zajednički jednoj i drugoj relaciji. Rezultat presjeka relacija *r* i *s* je skup svih slogova iz relacije *r* koji se nalaze u relaciji *s* ili obrnuto, pri čemu se duplikati slogova ne eliminiraju iz rezultata.

r ∩ s	A	B	C
	2	1	9
	8	1	3

6.1.3 Razlika

Oznaka operatora: **–**

Definicija razlike ostaje ista kao i kod klasične relacijske algebre, uz razliku da su relacije definirane kao multiskupovi zbog čega nema eliminacije duplikata slogova. Kardinalnost pojedinog sloga jednaka je razlici kardinalnosti sloga u svakoj od relacija. Ukoliko je kardinalnost sloga u relaciji s desne strane operacije veća od kardinalnosti tog sloga u lijevoj relaciji, tada je kardinalnost takvog sloga jednaka nuli.

Primjer 16. Skupovni operator razlika (*r* – *s*)

r	A	B	C
	3	1	7
	1	9	4
	2	2	8
	3	1	7

s	A	B	C
	3	1	7
	2	2	8
	6	4	1
	2	2	8

r – s – razlika relacija *r* i *s* je multiskup koji se sastoji od slogova iz prve relacije uz eliminaciju slogova koji su zajednički, odnosno slogova koji se pojavljuju u obje relacije, pri čemu se svaki od zajedničkih slogova eliminira onoliko puta kolika je minimalna kardinalnost tog sloga u nekoj od relacija.

r – s	A	B	C
	3	1	7
	1	9	4

6.2 Projekcija

Oznaka operatora: Π

Definicija projekcije ostaje ista kao i kod klasične relacijske algebre, uz razliku da su relacije definirane kao multiskupovi zbog čega nema eliminacije duplikata slogova. Osim toga broj n-torki u rezultantnoj relaciji jednak broju n-torki relacije na koju projekcija djeluje. U proširenoj relacijskoj algebri kao atributi na koje djeluje operator projekcije mogu se pronaći i algebarski izrazi. Projekcija implicitno definira operator preimenovanja, zbog čega se i operator preimenovanja eliminira iz proširene relacijske algebre.

Primjer 17. Projekcija ($\Pi_{A,C,D}(r)$)

r	A	B	C	D
	2	8	5	4
	1	9	5	6
	4	4	7	2
	7	4	2	8
	2	6	5	4

$\Pi_{A,C,D}(r)$ – projekcija relacije *r* je multiskup koji se sastoji od svih slogova iz relacije *r* uz promjenu relacijske sheme tog skupa. Relacijska shema rezultantne relacije jednaka je atributima koji su definirani projekcijom, a broj slogova rezultantne relacije jednak je broju slogova relacije *r* što znači da nema eliminacije duplikata. Duplikati se ne eliminiraju jer je relacija *r* definirana kao multiskup.

$\Pi_{A,C,D}(r)$	A	C	D
	2	5	4
	1	5	6
	4	7	2
	7	2	8
	2	5	4

Primjer 18. Projekcija ($\Pi_{A,C,B+D}(r)$)

r	A	B	C	D
	2	8	5	4
	1	9	5	6
	4	4	7	2
	7	4	2	8
	2	6	5	4

$\Pi_{A,C,B+D}(r)$	A	C	B+D
	2	5	12
	1	5	15
	4	7	6
	7	2	12
	2	5	10

Primjer 19. Projekcija ($\Pi_{A \longrightarrow E,B+D \longrightarrow Suma}(r)$)

r	A	B	C	D
	2	8	5	4
	1	9	5	6
	4	4	7	2
	7	4	2	8
	2	6	5	4

$\Pi_{A \rightarrow E,\ B+D \rightarrow Suma}(r)$	E	Suma
	2	12
	1	15
	4	6
	7	12
	2	10

6.3 Selekcija

Oznaka operatora: σ

Definicija selekcije ostaje ista kao i kod klasične relacijske algebre, uz razliku da su relacije definirane kao multiskupovi zbog čega nema eliminacije duplikata slogova.

Primjer 20. Selekcija ($\sigma_F(r)$)

r	A	B	C	D
t_1	1	a	2	b
t_2	2	b	1	a
t_3	9	a	7	c
t_4	7	c	4	a
t_5	8	a	5	c
t_6	9	a	7	c

Uvjet koji pojedini slog mora zadovoljiti definiran je s *F.*

$$F: (A > C) \wedge (D \neq a)$$

σ_F(r) – selekcija relacije *r* prema zadanom Boolean uvjetu *F* je multiskup koji se sastoji od svih slogova iz relacije *r* koji zadovoljavaju uvjet *F*.

σ_F(r)	A	B	C	D
t_3	9	a	7	c
t_5	8	a	5	c
t_6	9	a	7	c

t_1: $(1 > 2) \wedge (b \neq a) \equiv 0 \wedge 1 \equiv 0$

t_2: $(2 > 1) \wedge (a \neq a) \equiv 1 \wedge 0 \equiv 0$

t_3: $(9 > 7) \wedge (c \neq a) \equiv 1 \wedge 1 \equiv 1$

t_4: $(7 > 4) \wedge (a \neq a) \equiv 1 \wedge 0 \equiv 0$

t_5: $(8 > 5) \wedge (c \neq a) \equiv 1 \wedge 1 \equiv 1$

t_6: $(9 > 7) \wedge (c \neq a) \equiv 1 \wedge 1 \equiv 1$

6.4 Spojevi

Oznaka operatora: $\bowtie_{<uvjet\_povezivanja>}$

Spajanje relacija je vrlo važan dio svakog upitnog jezika pa tako i *SQL-a*. Na taj način je omogućeno postavljanje složenih upita koji se temelje na povezivanju većeg broja relacija. *SQL* jezik u tom pogledu nudi više vrsta različitih spojeva kojima se mogu ostvariti različite vrste povezivanja relacija.

Cijeli koncept povezivanja relacija kod proširene relacijske algebre temelji se na Θ spoju kod kojeg se koristi isti skup proširenih operatora kao i kod selekcije. Kako bi se pojedini istoimeni atributi relacija koje se spajaju razlikovali, uvodi se *alias*, odnosno oznaka kojom definiramo relaciju kojoj atribut pripada. Format *aliasa* je <*naziv_relacije*>.<naziv_atributa>, čime je definirana relacija :

atribut <naziv_atributa> pripada relaciji <*naziv_relacije*>

Za razliku od prirodnog spoja kod kojeg se relacije povezuju temeljem istoimenih atributa i njihove jednakosti, kod Θ spoja se definira uvjet prema kojem se određene relacije povezuju. Međutim, u radu s bazom podataka postoji više klauzula koje koristimo za povezivanje relacija. Način na koji smo do sada definirali Θ spoj predstavlja samo jednu vrstu spoja u *SQL-u*, koji se naziva *INNER JOIN*.

Osim najčešće korištenog *INNER JOIN-a* koji relacije spaja temeljem istih vrijednosti zajedničkih atributa, odnosno dohvaća sve slogove koji se podudaraju u zajedničkim atributima, ponekad se javlja potreba za korištenjem ostalih vrsta spojeva.

INNER JOIN spada u grupu unutarnjih spojeva, dok se spojevi koje ćemo spomenuti u nastavku ubrajaju u grupu vanjskih spojeva.

Razlikujemo 3 vrste vanjskih spojeva :

1. **LEFT JOIN** – dohvaća sve vrijednosti iz relacije s lijeve strane operacije uz pripadne vrijednosti relacije s desne strane operacije.
2. **RIGHT JOIN** – dohvaća sve vrijednosti iz relacije s desne strane operacije uz pripadne vrijednosti relacije s lijeve strane operacije.
3. **FULL (OUTER) JOIN** – dohvaća sve vrijednosti iz obje relacije. Ovaj spoj je amalgamacija LEFT JOIN-a te RIGHT JOIN-a.

Primjer 21. Theta spoj ($r \bowtie_{r.C = s.E} s$)

r	A	B	C
	2	1	1
	1	7	6
	9	2	4

s	D	E
	6	1
	1	1
	2	6

r $\bowtie_{r.C = s.E}$ **s** – theta spoj relacija *r* i *s* je multiskup koji se sastoji od kombinacija slogova iz relacija *r* sa slogovima iz relacije *s* koji imaju jednake vrijednosti atributa *r.*C i *s.*E.

r $\bowtie_{r.C = s.E}$ s	A	B	C	D	E
	2	1	1	6	1
	2	1	1	1	1
	1	7	6	6	2

U ovom primjeru možemo vidjeti razliku između prirodnog spoja klasične relacijske algebre te spojeva u proširenoj relacijskoj algebri. Ključna razlika je u uvjetu koji u proširenoj relacijskoj algebri definira atribute na temelju kojih se relacije povezuju, a kojeg kod prirodnog spoja nema.

Primjer 22. Lijevo vanjsko spajanje ($r \bowtie_{r.C = s.C} s$) – LEFT JOIN

r	A	B	C
	c	2	5
	d	1	9
	a	6	7

s	C	D	E
	5	a	8
	7	c	3
	1	b	4

r $\bowtie_{r.C = s.C}$ **s** – lijevo vanjsko spajanje relacija *r* i *s* je multiskup koji se sastoji od kombinacija slogova iz prve relacije sa slogovima iz druge relacije koje imaju iste vrijednosti zajedničkih atributa uz dodavanje slogova relacije s lijeve strane operacije koji „ne sudjeluju“ u spajanju.

r $\bowtie_{r.C = s.C}$ s	A	B	C	D	E
	c	2	5	a	8
	a	6	7	c	3
	d	1	9	*NULL*	*NULL*

U rezultantnoj relaciji možemo primijetiti slog s vrijednostima atributa *NULL*. Razlog tome je što se rezultantna relacija sastoji od kombinacija slogova iz prve relacije sa slogovima iz druge relacije koje imaju iste vrijednosti zajedničkih atributa uz dodavanje slogova relacije s lijeve strane operacije koji „ne sudjeluju“ u spajanju. Naglasak je na slogovima relacije koja se nalazi s lijeve strane operacije, a koji „ne sudjeluju“ u spajanju. Slog *t2* relacije *r*, odnosno slog čije su vrijednosti atributa A, B, C jednake d, 1, 9 respektivno, nema zajedničku vrijednost atributa C s relacijom *s* te se iz tog razloga ne mogu spojiti. Kako je ovdje riječ o *LEFT JOIN-u* taj slog mora biti uključen u rezultantnu relaciju jer je dio relacije koja se nalazi s lijeve strane operacije, pa će vrijednosti atributa D i E biti *NULL*.

Primjer 23. Desno vanjsko spajanje ($r \rtimes_{r.C = s.C} s$) – RIGHT JOIN

r	A	B	C
	c	2	5
	d	1	9
	a	6	7

s	C	D	E
	5	a	8
	7	c	3
	1	b	4

r $\rtimes_{r.C = s.C}$ **s** – desno vanjsko spajanje relacija *r* i *s* je multiskup koji se sastoji od kombinacija slogova iz prve relacije sa slogovima iz druge relacije koje imaju iste vrijednosti zajedničkih atributa uz dodavanje slogova relacije s desne strane operacije koji „ne sudjeluju“ u spajanju.

r $\rtimes_{r.C = s.C}$ s	A	B	C	D	E
	c	2	5	a	8
	a	6	7	c	3
	NULL	*NULL*	1	b	4

U rezultantnoj relaciji možemo primijetiti slog s vrijednostima atributa *NULL*. Razlog tome je što se rezultantna relacija sastoji od kombinacija slogova iz prve relacije sa slogovima iz druge relacije koje imaju iste vrijednosti zajedničkih atributa uz dodavanje slogova relacije s desne strane operacije koji „ne sudjeluju“ u spajanju. Naglasak je na slogovima relacije koja se nalazi s desne strane operacije, a koji „ne sudjeluju“ u spajanju. Slog *t3* relacije *s*, odnosno slog čije su vrijednosti atributa C, D, E jednake 1, b, 4 respektivno, nema zajedničku vrijednost atributa C s relacijom *r* te se iz tog razloga ne mogu spojiti. Kako je ovdje riječ o *RIGHT JOIN-u* taj slog mora biti uključen u rezultantnu relaciju jer je dio relacije koja se nalazi s desne strane operacije, pa će vrijednosti atributa A i B biti *NULL*.

Primjer 24. Potpuno vanjsko spajanje ($r \bowtie_{r.C = s.C} s$) – FULL JOIN

r	A	B	C
	c	2	5
	d	1	9
	a	6	7

s	C	D	E
	5	a	8
	7	c	3
	1	b	4

r $\bowtie_{r.C = s.C}$ **s** – potpuno vanjsko spajanje relacija *r* i *s* je multiskup koji se sastoji od kombinacija slogova iz prve relacije sa slogovima iz druge relacije koje imaju iste vrijednosti zajedničkih atributa uz dodavanje slogova koji „ne sudjeluju“ u spajanju.

Vrijedi da je **r** $\bowtie_{r.C = s.C}$ **s = *r*** $\bowtie_{r.C = s.C}$ ***s*** **∪ *r*** $\bowtie_{r.C = s.C}$ ***s***.

r $\bowtie_{r.C = s.C}$ s	A	B	C	D	E
	C	2	5	a	8
	A	6	7	c	3
	D	1	9	*NULL*	*NULL*
	NULL	*NULL*	1	b	4

U rezultantnoj relaciji možemo primijetiti slogove s vrijednostima atributa *NULL*. Razlog tome je što se rezultantna relacija sastoji od kombinacija slogova iz prve relacije sa slogovima iz druge relacije koje imaju iste vrijednosti zajedničkih atributa uz dodavanje slogova koji „ne sudjeluju“ u spajanju. Naglasak je na slogovima relacije koji „ne sudjeluju“ u spajanju. Slog *t2* relacije *r*, odnosno slog čije su vrijednosti atributa A, B, C jednake d, 1, 9 respektivno, nema zajedničku vrijednost atributa C s relacijom *s* te se iz tog razloga ne mogu spojiti. Također i slog *t3* relacije *s*, odnosno slog čije su vrijednosti atributa C, D, E jednake 1, b, 4 respektivno, nema zajedničku vrijednost atributa C s relacijom *r* te se iz tog razloga ne mogu spojiti. Kako je ovdje riječ o *FULL OUTER JOIN-u* ti slogovi moraju biti uključen u rezultantnu relaciju pa će vrijednosti atributa A i B za slog *t4* te D i E za slog *t3* biti *NULL*.

6.5 Produkt (Kartezijev produkt)

Oznaka operatora: ⊗

Definicija Kartezijevog produkta ostaje ista kao i kod klasične relacijske algebre, uz razliku da su relacije definirane kao multiskupovi zbog čega nema eliminacije duplikata slogova. U terminologiji proširene relacijske algebre Kartezijev produkt možemo promatrati kao Θ spoj bez definiranih uvjeta povezivanja relacija. *SQL* standard ima definiranu klauzulu CROSS JOIN koja reprezentira Kartezijev produkt. Zbog kvadratne prostorne složenosti preporučuje se izbjegavati Kartezijev produkt kad god je to moguće.

Primjer 25. Kartezijev produkt (*r* ⊗ *s*)

r	A	B	C	D
	6	a	B	4
	3	c	B	1
	9	b	A	2

s	B	C	E
	a	b	2
	a	b	9
	c	a	7

r ⊗ s – Kartezijev produkt relacija *r* i *s* je multiskup koji se sastoji od svih kombinacija slogova iz prve relacije sa slogovima iz druge relacije pri čemu je potrebno preimenovati zajedničke atribute ukoliko relacijske sheme nisu međusobno disjunktne.
Duplikati slogova se ne eliminiraju iz rezultantne relacije.

r ⊗ s	A	r.B	r.C	D	s.B	s.C	E
	6	a	b	4	a	b	2
	6	a	b	4	a	b	9
	6	a	b	4	c	a	7
	3	c	b	1	a	b	2
	3	c	b	1	a	b	9
	3	c	b	1	c	a	7
	9	b	a	2	a	b	2
	9	b	a	2	a	b	9
	9	b	a	2	c	a	7

6.6 Operator eliminacije duplikata

Oznaka operatora: δ

Proširena relacijska algebra, kao i *SQL* upitni jezik koristi multiset semantiku što znači da se pojedini slogovi mogu pojaviti više puta u relacijama. Takvi slogovi nisu od prevelikog značaja zato što pružaju istovjetne informacije, pa je duplikate slogova poželjno ukloniti.
Cilj nam je postići jedinstvenost slogova relacije, a ujedno i smanjiti broj slogova relacije, jer duplikati slogova nepotrebno povećavaju relaciju.
Operator eliminacije duplikata je unarni operator koji vraća sve slogove, odnosno n-torke relacije, ali bez duplikata.
Ekvivalentan je klauzuli *DISTINCT* iz *SELECT* naredbe te je dio proširene relacijske algebre.

Primjer 26. Operator eliminacije duplikata ($\delta(r)$)

r	A	B	C
	1	2	3
	1	2	3
	5	9	3
	6	4	2
	5	9	3

$\delta(r)$ – iz relacije *r* eliminiramo sve slogove koji se javljaju dva ili više puta, odnosno eliminiramo sve duplikate slogova. Na taj način se ostvaruje jedinstvenost slogova relacije koja značajno doprinosi optimizaciji vremena izvođenja upita o kojoj ćemo kasnije govoriti.

δ(r)	A	B	C
	1	2	3
	5	9	3
	6	4	2

6.7 Operator grupiranja

Oznaka operatora: γ

Kako bi zadovoljila zahtjeve *SQL-a*, proširena relacijska algebra uvodi operator grupiranja γ, koji se koristi za grupiranje i agregiranje slogova relacije.
Operator grupiranja je unarni operator koji grupira vrijednosti relacije prema određenom kriteriju te vraća relaciju kao rezultat.
Operator grupiranja se uobičajeno kombinira s drugim operatorima relacijske algebre čime se omogućava grupiranje prema određenom kriteriju, grupiranje agregiranih slogova i slično.
Razlikujemo 5 agregacijskih funkcija koje su definirane u svakom upitnom jeziku :

1. **avg** – vraća prosječnu vrijednost specificiranog atributa relacije.
2. **min** – vraća minimalnu vrijednost specificiranog atributa relacije.
3. **max** – vraća maksimalnu vrijednost specificiranog atributa relacije.
4. **sum** – vraća sumu vrijednosti specificiranog atributa relacije.
5. **count** – vraća broj slogova relacije.

Sintaksa operatora grupiranja je sljedeća.

$$_{G1, G2, ..., Gn}\gamma_{F(A)}(r),$$

gdje je

- G1, G2, ... , Gn lista atributa po kojima se vrši grupiranje. Lista atributa po kojima se vrši grupiranje može biti prazna, pri čemu se relacija interpretira kao jedna grupa
- F je agregacijska funkcija prema kojoj će podaci biti agregirani
- A je atribut relacije čije vrijednosti će biti agregirane

Prilikom agregiranja atributa, u rezultantnoj relaciji taj atribut dobiva naziv temeljem naziv agregacijske funkcije koja je korištena u operaciji. Kako bi to izbjegli, agregiranje podataka, se uobičajeno kombinira s preimenovanja atributa.
Ako se operator koristi samo za grupiranje slogova tada je njegova sintaksa sljedeća :

$$\gamma_L(r),$$

gdje je L lista atributa relacije *r* prema kojima se vrši grupiranje.

U *SQL-u* se grupiranje obično vrši prema nekom kriteriju za što se koristi klauzula *HAVING*. Kako je *HAVING* klauzula poopćenje *WHERE* klauzule, u terminologiji relacijske algebre klauzula *HAVING* je ekvivalentna operatoru selekcije. Vidljivo je da samostalno korištenje operatora γ nema veliku primjenu te se iz tog razloga obično kombinira s agregacijskim funkcijama kao što smo ranije naveli.

Primjer 27. Operator grupiranja ($\Pi_{\text{VlasnikRacuna, sum(StanjeRacuna)}}(_{\text{VlasnikRacuna}}\gamma_{\text{sum(StanjeRacuna)}}(r))$)

r	VlasnikRacuna	BrojRacuna	StanjeRacuna
	Pero Perić	HR4224020062763457298	4300
	Ana Anić	HR0623400096162152877	3900
	Lara Larić	HR7824020064237488632	1450
	Pero Perić	HR5625000095933515242	2150
	Lara Larić	HR1423400093415511888	1600

$\Pi_{\text{VlasnikRacuna,sum(StanjeRacuna)}}(_{\text{VlasnikRacuna}}\gamma_{\text{sum(StanjeRacuna)}}(r))$– agregacija relacije *r* prema agregacijskoj funkciji *sum* je skup koji se sastoji od slogova relacije *r* grupiranih prema vlasniku računa te sume stanja **svih** računa svakog korisnika.

$s = \Pi_{\text{VlasnikRacuna,sum(StanjeRacuna)}}(_{\text{VlasnikRacuna}}\gamma_{\text{sum(StanjeRacuna)}}(r))$

s	VlasnikRacuna	sum(StanjeRacuna)
	Pero Perić	6450
	Ana Anić	3900
	Lara Larić	3050

6.8 Operator sortiranja

Oznaka operatora: τ

Operator sortiranja je unarni operator koji se, kao što i sam naziv kaže, koristi za sortiranje slogova relacije. Sortiranje se uobičajeno koristi za lakše pronalaženje podataka te organizaciju podataka. Operator sortiranja ne mijenja relacijsku shemu relacije na koju se primjenjuje.

Sintaksa operatora sortiranja :

$$\tau_{L<ASC \mid DESC>}(r),$$

gdje je L lista atributa po kojim se vrši sortiranje, *ASC* uzlazno sortiranje, a *DESC* silazno sortiranje.

Primjer 28. Operator sortiranja ($\tau_{Ime,\ PrezimeASC}(r)$)

r	Ime	Prezime	EmailAdresa
	Pero	Perić	pperic@foi.hr
	Ana	Anić	aanic1@foi.hr
	Lara	Larić	lariclana@foi.hr
	Karlo	Karlić	karlicka4@foi.hr
	Marko	Markić	markicmarko1@foi.hr

$\tau_{Ime,\ PrezimeASC}(r)$ – rezultat sortiranja relacije *r* je lista koja se sastoji od svih slogova relacije *r*, bez promjene relacijske sheme relacije *r*, sortiranih prema imenu i prezimenu. Sortiranje se primarno izvršava prema imenu, a ukoliko se pojavi više slogova s istom vrijednošću imena, tada se takvi slogovi dodatno sortiraju uzlazno prema prezimenu.

$\tau_{Ime,\ PrezimeASC}(r)$	Ime	Prezime	EmailAdresa
	Ana	Anić	aanic1@foi.hr
	Karlo	Karlić	karlicka4@foi.hr
	Lara	Larić	lariclana@foi.hr
	Marko	Markić	markicmarko1@foi.hr
	Pero	Perić	pperic@foi.hr

7. Prevođenje SQL upita

Kako bi cijeli postupak pretvorbe bio jasniji, prvo ćemo reći nešto o samom *SQL-u*. *SQL* jezik je najčešće korišten jezik za rad s relacijskim bazama podataka. Kratica *SQL* dolazi od ***S****tructured* ***Q****uery* ***L****anguage*, što bi značilo strukturirani upitni jezik.

„U skladu sa svojim imenom, *SQL* u prvom redu omogućava postavljanje jednostavnih i složenih upita u relacijskim bazama podataka.“ (*Baze podataka*, Manger R., 2012. , 84. str.)

U terminologiji baza podataka, upiti ne predstavljaju samo nužne izraze koji se koriste za dohvaćanje podataka iz baza podataka. *SQL* kao jezik je u tom smislu daleko napredniji i razvijeniji te omogućava manipulaciju s bazom podataka na način da osigurava naredbe za kreiranje relacija (*CREATE TABLE*), unos podataka (*INSERT*), ažuriranje podataka (*UPDATE*), brisanje podataka (*DELETE*), upravljanje transakcijama (*TRANSACTION*), dodjelu (*GRANT*) te oduzimanje (*REVOKE*) ovlaštenja i sl. Važno je napomenuti kako *SQL* ima ugrađene funkcije za operacije nad podacima.

„Upitni dio SQL-a uglavnom je zasnovan na relacijskom računu, s time da je matematička notacija zamijenjena ključnim riječima nalik na engleski govorni jezik.“

(*Baze podataka*, Manger R., 2012. , 84. str.)

Zbog svojih svojstava, *SQL* je u semantičkom smislu ekvivalentan relacijskoj algebri što omogućava vrlo jednostavnu pretvorbu u upit relacijske algebre. Kako se radi o ekvivalentnim operacijama pretvorbe vrijedi i obrat, odnosno svaki upit relacijske algebre se može vrlo jednostavno pretvoriti u korespondentan *SQL* upit. *SQL* je u tom smislu deklarativni jezik (što), dok je relacijska algebra proceduralni jezik (kako).

Vrlo je važno da se prilikom pretvorbe pridržavamo prioriteta pojedinog operatora relacijske algebre.

Prioritet operatora od najvećeg prema najmanjem je redom:

1. projekcija
2. selekcija
3. produkt
4. prirodno spajanje / kvocijent
5. razlika
6. presjek / unija

Logičke operacije, *NOT, AND* i *OR*, također imaju definiran prioritet. Prioritet logičkih operatora od najvećeg prema najmanjem je redom :

1. NOT
2. AND
3. OR

Ekvivalencija između *SQL* klauzula i operatora relacijske algebre je definirana na sljedeći način. Klauzula *SELECT* je ekvivalentna projekciji (⊓), *FROM* je ekvivalentna Kartezijevom produktu (⊗), *WHERE/JOIN* je ekvivalentna selekciji (σ), *UNION* je ekvivalentna uniji (∪), *INTERSECT* je ekvivalentna presjeku (∩), *EXCEPT/MINUS* je ekvivalentna razlici (–), *GROUP BY* je ekvivalentna operatoru grupiranja (γ), *ORDER BY* je ekvivalentna operatoru sortiranja (τ), *DISTINCT* je ekvivalentna eliminaciji duplikata (δ),

Do sada smo se bavili teorijskim aspektima relacijske algebre, *SQL-a* te upita.
Pretpostavimo da imamo zadan *SQL* upit i relacije *r* i *s*.
SQL(U) :

SELECT	A, C, E, F
FROM	r, s
WHERE	(C = D)
AND	(A = 1)
AND	(B = 'a')

r	A	B	C
	1	a	7
	1	b	7
	2	f	4
	5	h	2
	1	a	1

s	D	E	F
	9	2	c
	7	3	d
	1	9	a
	4	1	e

Jasno je da će ovaj *SQL* rezultirati odgovorom čija relacija sadrži slog ili slogove koji zadovoljavaju sva ograničenja koja su definirana upitom.

◦U	A	C	E	F
	1	7	3	d
	1	1	9	a

Postavlja se pitanje:
„Na koji način se izvršava upit, odnosno što se događa u pozadini *SUBP-a*?“

7.1 Nevezani podupit

Za upit kažemo da je nevezani ako se u njemu ne nalaze neki od atributa relacija iz glavnog upita. Nevezani upit se obično javlja prilikom korištenja klauzula *IN, NOT IN, ANY* te *ALL* te podupita koji kao odgovor vraćaju jedan slog, dok se vrlo rijetko, mogli bismo reći gotovo nikad, ne javljaju s klauzulom *EXISTS* jer u takvim slučajevima nemaju smisla.
Svaki nevezani upit izvršava se neovisno od glavnog upita te se na temelju dobivenog rezultata podupita kreira privremena relacija koja zatim sudjeluje u izvršavanju glavnog upita.
Izvršavanje podupita može biti uključeno u glavni upit, te u tom slučaju nije potrebna privremena relacija za pohranu rezultata podupita.

Primjer 29. Nevezani upit (*Obrada upita* (prezentacija), 2019. , A. Lovrenčić)
SQL(U) :

SELECT	Prezime, Ime
FROM	osoba
WHERE	OsobaId
IN	(SELECT Osoba FROM predavac)

Prvo je potrebno rezultat podupita pohraniti u privremenu relaciju koja se zatim koristi kod izvršavanja glavnog upita.

$$r = \delta(\Pi_{Osoba}(\text{predavac}))$$

Nakon što se taj upit izvrši, relacija *r* se koristi prilikom izvršavanja glavnog upita.

$$\Pi_{Prezime,\ Ime}(\textit{osoba} \bowtie r)$$

7.2 Vezani podupit

Za upit kažemo da je vezani ako se u njemu nalaze neki od atributa relacija iz glavnog upita. Vezani upit se obično javlja prilikom korištenja klauzula *IN, NOT IN, ANY, ALL* te *EXISTS*. Kod vezanih upita *EXISTS* ima smisla te se iz tog razloga vrlo često koristi.
Ako je podupit vezan za glavni upit, tada se iz njegove *WHERE* klauzule brišu svi uvjeti koji su vezani za glavni upit. Nakon toga se takav modificirani podupit izvršava te sprema u pomoćnu relaciju. Nakon dobivanja rezultata podupita, privremena relacija se povezuje s glavnim upitom korištenjem Θ spoja, odnosno operatora ⋈.
Važno je napomenuti da se kod povezivanja dodaju i uvjeti koji su vezani uz glavni upit.

Primjer 30. Vezani upit (*Obrada upita* (prezentacija), 2019. , A. Lovrenčić)
SQL(U) :

```
SELECT      Prezime, Ime
FROM        osoba
WHERE       OsobaId
IN          (SELECT Osoba FROM ispit
            WHERE Datum < DatumRodjenja)
```

Prvo je potrebno rezultat podupita pohraniti u privremenu relaciju koja se zatim koristi kod izvršavanja glavnog upita.

$$r = \delta(\Pi_{\text{Osoba, DatumRodjenja}}(\text{ispit}))$$

Nakon što se taj upit izvrši, relacija *r* se koristi prilikom izvršavanja glavnog upita.

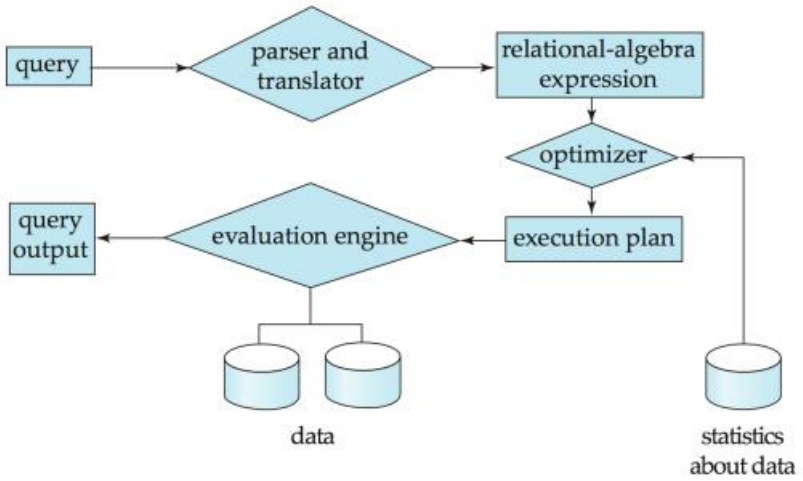
$$\sigma_{\text{Datum} < \text{DatumRodjenja}}(\Pi_{\text{Prezime, Ime}}(\textit{osoba} \bowtie r))$$

O podupitima bi se moglo daleko više govoriti, ali mi ćemo u okviru relacijske algebre stati na ovome. Važno je samo napomenuti da ukoliko upit sadrži podupite, tada se takav *SQL* upit prevodi u više zasebnih upita u proširenoj relacijskoj algebri koji se zatim i zasebno izvršavaju. Prvo se izvršava svaki od podupita, redoslijedom kojim je određeno, a tek onda glavni upit.

7.3 Procesor upita

Procesor upita se koristi za prevođenje te izvršavanje definiranog *SQL* upita. U osnovi se sastoji od dvije komponente :

1. *query compiler*
2. *execution (evaluation) engine*



Slika 1. Procesor upita (Rasmus Ejlers Møgelberg, 2012, 9. slajd)

Dakle, postupak dobivanja odgovora na postavljeni upit nije tako jednostavan kao što se ranije činilo. Nakon postavljanja upita, isti je potrebno parsirati i prevesti u sintaksu relacijske algebre za što je zadužen *parser*, odnosno *query preprocessor*. *Parser* je komponenta *SUBP-a* koja je zadužena za generiranje stabla parsiranja koje *query preprocessor* zatim koristi za prijevod u upit relacijske algebre. Taj proces je poznat pod nazivom semantička analiza. Ukoliko *parser* uspije izgraditi stablo parsiranja tada je upitni izraz ispravno definiran, dok u suprotnom upitni izraz nije ispravno definiran što je vidljivo porukom o sintaktičkoj grešci koju prikazuje *SUBP*. Nakon što je *parser* uspješno konstruirao stablo parsiranja tada dolazi na red semantička provjera definiranog upita. Ukoliko nema semantičkih grešaka tada se prelazi na fazu prevođenja upita. *Query preprocessor* na temelju konstruiranog stabla parsiranja kreira stablo algebarskih operatora koje reprezentira upit u relacijskoj algebri, odnosno logički plan upita koji se zatim optimizira. *Optimizer* komponenta dobiveni upit u relacijskoj algebri optimizira korištenjem pravila optimizacije upita koje ćemo posebno obraditi u nastavku. Optimizacija se obavlja na logičkoj i fizičkoj razini. Nakon optimizacije, upit se izvršava prema definiranom planu izvršavanja te dobivamo odgovor na postavljeni upit. Možemo reći da je kreiranja stabla parsiranja najvažniji korak prilikom pretvorbe *SQL* upita u upit relacijske algebre.

7.3.1 Query compiler

Query compiler je komponenta procesora upita koja je zadužena za izradu logičkog upitnog plana. Upitni plan predstavlja set operacija koje će biti primijenjene nad podacima, a koje su pohranjenje u obliku svojevrsnog plana.
„Glavni dio *query compiler-a* čine tri jedinice :

1. *query parser* koji je zadužen za kreiranja stabla parsiranja, o kojem ćemo detaljnije govoriti u nastavku
2. *query preprocessor* koji je zadužen za semantičku provjeru definiranog upita, odnosno utvrđivanja da relacije koje se koriste u izrazu zaista postoje te transformaciju stabla parsiranja u stablo algebarskih operatora koje reprezentira inicijalni, odnosno logički upitni plan
3. *query optimizer* koji je zadužen za transformaciju inicijalnog upitnog plana u najbolju moguću sekvencu operacija koje će se izvesti nad podacima“

(prema *Database System The Complete Book*, Garcia Molina H. i suradnici, 2009. , 10. str.)

Query compiler koristi određene metapodatke te podatkovne statističke analize kako bi utvrdio koji slijed operacija je optimalan za izvršavanje. Primjerice, ukoliko određena relacija ima definiranu pomoćnu strukturu podataka (npr. *index*) tada izvršavanje takvog plana može biti zamjetno brže od nekog drugog.

7.3.2 Execution engine

Execution engine je komponenta procesora upita koja je zadužena za izvršavanje svakog koraka fizičkog upitnog plana.
Execution engine neprestano komunicira s većinom drugih komponenata koje sačinjavaju *SUBP*, bilo direktno ili pomoću deklariranih međuspremnika.
Kako bi manipulirao nad podacima, prvo ih mora iz baze podataka dohvatiti u međuspremnike. Važno je napomenuti da *execution engine* neprestano komunicira sa *scheduler-om*, kako bi izbjegao iznimke uzrokovane pristupom podacima koji su zaključani te *log managerom* kako bi ustanovio da su sve promjene nad bazom uspješno odrađene.

7.4 Gramatika SQL upitnog jezika

Kako bi mogli definirati gramatiku *SQL* upitnog jezika, prvo moramo reći nešto o formalnoj gramatici upitnih jezika. Gramatika se koristi za provjeru sintaktičke korektnosti upita. „Gramatika upitnih jezika je sustav koji se sastoji od konačnog broja *terminal* elemenata, konačnog broja *non-terminal* elemenata i konačnog broja *produkcijskih pravila.*“
(*Relational Algebra Expression Evaluation*, L. Malkova, 2009. , 19. str.)
„Terminal elementi predstavljaju elemente od kojih se grade upitni izrazi nekog jezika što znači da se ti elementi ne mogu generalizirati, odnosno detaljnije specificirati. *Terminal* elementi su elementi na najnižoj razini složenosti.
Non-terminal elementi predstavljaju elemente koji se koriste za izricanje semantike upitnih izraza te se kao takvi nikad direktno ne pojavljuju u nekom upitnom izrazu.
Produkcijska pravila su neprazni skupovi *terminal* elemenata i *non-terminal* elemenata na lijevoj strani nakon čega slijedi proizvoljan skup *non-terminal* elemenata i *terminal* elemenata na desnoj strani. Koriste se za definiranje vrijednosti *non-terminal* elemenata, odnosno svaki simbol s lijeve strane produkcijskog pravila može se zamijeniti s nekim simbolom s desne strane produkcijskog pravila.
Početni simbol ubraja se u *non-terminal* elemente te predstavlja oznaku od koje počinje generiranje upitnog izraza. Početni simbol mora uvijek biti definiran jer u suprotnom gramatika nekog upitnog jezika nije u potpunosti definirana.“
(prema *Relational Algebra Expression Evaluation*, L. Malkova, 2009. , 20. str.)
SQL koristi, tzv. *gramatiku bez konteksta* koja se može eksplicitno koristiti unutar *parser-a*. Gramatika bez konteksta je vrsta gramatike u kojoj se na lijevoj strani produkcijskog pravila pojavljuju samo *non-terminal* elementi.
U kontekstu gramatike *SQL-a* koriste se sljedeći simboli :

- simbol **:** označava s kojim *terminal* elementima s desne strane može biti zamijenjen *non-terminal* element s lijeve strane produkcijskog pravila
- izrazi koji su definirani malim slovima označavaju *non-terminal* elemente, dok izrazi koji su definirani velikim slovima označavaju *terminal* elemente. U kontekstu relacijske algebre možemo reći da *terminal* elementi predstavljaju operatore relacijske algebre koji će biti korišteni unutar upitnih izraza
- simbol **|** označava da se *non-terminal* element može zamijeniti s više *terminal* elemenata koji su zatim definirani u nastavku produkcijskog pravila
- simbol **;** se koristi za terminiranje produkcijskog pravila

Ukoliko početni simbol nije definiran tada se kao početni simbol podrazumijeva prvi *non-terminal* element. U nastavku ćemo definirani produkcijska pravila *SELECT* naredbe.

7.4.1 Produkcijska pravila SELECT naredbe

<query> :

Q1 <query> ← <simple_query>

Q2 | <query> ← (<query>)

Q3 | <query> ← <query> <set_op> <query>

;

<set_op> :

SO1 <set_op> ← <set_comp>

SO2 | <set_op> ← <set_comp> ALL

;

<set_comp> :

SC1 <set_comp> ← UNION

SC2 | <set_comp> ← INTERSECT

SC3 | <set_comp> ← EXCEPT

;

<simple_query> :

SQ1 <simple_query> ← <SFW>

SQ2 | <simple_query> ← <SFW> <gho_clouse>

;

<SFW> :

SFW1 <SFW> ← SELECT <select_list> FROM <from_list>

SFW2 | <SFW> ← SELECT <select_list> FROM <from_list> WHERE <condition>

;

<select_list> :

SL1 <select_list> ← <select_attr_list>

SL2 | <select_list> ← DISTINCT <select_attr_list>

;

<select_attr_list> :

SAL1 <select_attr_list> ← *

SAL2 | <select_attr_list> ← <project_list>

;

<project_list> :

PL1 <project_list> ← <attr>

PL2 | <project_list> ← <attr> , <project_list>

;

<attr> :

AT1 <attr> ← <attr_def>

AT2 | <attr> ← <alias> . <attr_def>

;

<attr_def> :

AD1 <attr_def> ← <attr_name>

AD2 | <attr_def> ← <group_function>(<attr_def>)

AD3 | <attr_def> ← COUNT(*)

AD4 | <attr_def> ← <expression>

;

<attr_name> :

ATN1 <attr_name> ← <attribute>

ATN2 | <attr_name> ← <table_name> . <attribute>

ATN3 | <attr_name> ← <alias> . <attribute>

;

<group_function> :

GF1 <group_function> ← COUNT

GF2 | <group_function> ← SUM

GF3 | <group_function> ← AVG

GF4 | <group_function> ← MAX

GF5 | <group_function> ← MIN

;

<from_list> :

FL1 <from_list> ← <table_reference>

FL2 | <from_list> ← <join_clause>

FL3 | <from_list> ← < table_reference> , <from_list>

;

<table_reference> :

TR1 <table_reference> ← <table_name>

TR2 | <table_reference> ← < table_name > AS <alias>

;

<table_name> :

TN1 <table_name> ← <table>

TN2 | <table_name> ← <table_space> . <table>

;

<join_clause> :

JC1 <join_clause> ← <join_table_clause> ON <join_column_clause>

;

<join_table_clause> :

JTC1 <join_table_clause> ← <table_name> <join_type> <table_name>

JTC2 | <join_table_clause> ← <table_name> <join_type> <join_clause>

;

<join_type> :

JT1 <join_type> ← JOIN

JT2 | <join_type> ← <obligation_type> JOIN

;

<join_column_clause> :

JCC1 <join_column_clause> ← <join_type> <table_name>

JCC2 | <join_column_clause> ← <join_type> <join_ clause>

JCC3 | <join_column_clause> ← <attr_name> <relational_operator> <attr_name>

;

<obligation_type> :

OT1 <obligation_type> ← CROSS

OT2 | <obligation_type> ← FULL

OT3 | <obligation_type> ← LEFT

OT4 | <obligation_type> ← RIGHT

;

<condition> :

CN1 <condition> ← <single_condition>

CN2 | <condition> ← (<condition>)

CN3 | <condition> ← NOT <condition>

CN4 | <condition> ← <single_condition> <logical_operator> <condition>

;

<logical_operator> :

LO1 <logical_operator> ← AND

LO2 | <logical_operator> ← OR

LO3 | <logical_operator> ← NOT

;

<single_condition> :

SCN1 <single_condition> ← <expression> <relational_operator> <expression>

SCN2 | <single_condition> ← <expression> BETWEEN <expression> AND <expression>

SCN3 | <single_condition> ← <expression> LIKE <expression>

SCN4 | <single_condition> ← <expression> LIKE <regular_expression>

SCN5 | <single_condition> ← <subquery>

SCN6 | <single_condition> ← left_arrow <expression> IS <NULL_condition>

;

<NULL_condition> :

NC1 <NULL_condition> ← NULL

NC2 | <NULL_condition> ← NOT NULL

;

<relational_operator> :

RO1 <relational_operator> ← <>

RO2 | <relational_operator> ← =

RO3 | <relational_operator> ← <

RO4 | <relational_operator> ← <=

RO5 | <relational_operator> ← >

RO6 | <relational_operator> ← >=

;

<subquery> :

SQ1 <subquery> ← <expression> IN (<query>)

SQ2 | <subquery> ← <expression> NOT IN (<query>)

SQ3 | <subquery> ← EXISTS (<query>)

SQ4 <subquery> ← NOT EXISTS (<query>)

SQ5 | <subquery> ← <expression> <relative_operator> ANY (<query>)

SQ6 | <subquery> ← <expression> <relative_operator> ALL (<query>)

;

<gho_clause> :

GHC1 <gho_clause> ← <ho_clause>

GHC2 | <gho_clause> ← GROUP BY <attribute_list>

GHC3 | <gho_clause> ← GROUP BY <attribute_list> <ho_clause>

;

<ho_clause> :

HC1 <ho_clause> ← <order_clause>

HC2 | <ho_clause> ← HAVING <having_condition>

HC3 | <ho_clause> ← HAVING <having_condition> <order_clause>

;

<attribute_list> :

AL1 <attribute_list> ← <attribute>

AL2 | <attribute_list> ← <attribute> , <attribute_list>

;

<single_having_condition> :

SHC1 <single_having_condition> ← <group_function> <relational_operator> <expression>

SHC2 | <single_having_condition> ← <group_function> BETWEEN <expression> AND <expression>

SHC3 | <single_having_condition> ← left_arrow <group_function> IS <NULL_condition>

;

<order_clause> :

OC1 <order_clause> ← ORDER BY <order_list>

;

<order_list> :

OL1 <order_list> ← <order_element>

OL2 | <order_list> ← <order_element> , <order_list>

;

<order_element> :

OE1 <order_element> ← <order_name>

OE2 | <order_element> ← <order_name> ASC

OE3 | <order_element> ← <order_name> DESC

;

<order_name> :

ON1 <order_name> ← <attribute_name>

ON2 | <order_name> ← <alias> . <attribute_name>

;

(*Obrada upita* (prezentacija), 2019. , A. Lovrenčić)

Jasno je da nema smisla definirati prazni upitni izraz što i gramatika relacijske algebre ne dozvoljava. Najjednostavniji izraz koji se može definirati i koji je dopušten gramatikom je naziv relacije.

U tom slučaju će *non-terminal* element biti zamijenjen s *terminal* elementom *table_reference*, *table_reference* sa *table_name* da bi u konačnici *table_name* bio zamijenjen s konkretnim nazivom relacije. Rezultat takvog izraza sadržavao bio sve n-torke definirane relacije.

Gramatika dopušta korištenje aritmetičkih te logičkih operatora. Unutar gramatike definiran je i format liste atributa za projekciju te format liste uvjeta za selekciju.
Svaki izraz može se obaviti oblim zagradama čime se postiže prednost djelovanja nekog operatora, ali i strukturiranje samog izraza kako bi bio lakši za razumijevanje.
„Prilikom korištenja binarnih operatora koristi se *infix* notacija kod koje se operator upisuje između operanada, dok se prilikom korištenja unarnih operatora koristi *prefix* notacija kod koje operator prethodi operandu.“
(prema *Relational Algebra Expression Evaluation*, L. Malkova, 2009. , 21. str.)

Uobičajeno je da se standardna sintaksa relacijske algebre, zbog svoje nepraktičnosti, zamijeni ekvivalentom sintaksom koja je lako pamtljiva i jasno definirana. Nepraktičnost upotrebe standardne sintakse očituje se u operatorima, čiji su simboli definirani grčkim alfabetom, koje je vrlo teško „pisati“ korištenjem standardne tipkovnice.

Operator	RA	Proposed Syntax
Union	$\cup$	$r_1 + r_2$
Intersection	$\cap$	$r_1 \ \hat{} \ r_2$
Set difference	$-$	$r_1 - r_2$
Natural join	$\bowtie$	r_1 @ r_2
Left join	$\rightthreetimes$	$r_1 \sim$ @ r_2
Right join	$\leftthreetimes$	r_1 @ $\sim r_2$
Cartesian product	$\times$	$r_1 * r_2$
Selection	σ	\$ [condition] (r_1)
Projection	π	# [list of attributes] (r_1)
Relation Rename	ρ	% $[r_2]$ (r_1)
Relation and Attributes Rename	ρ	% $[r_2$ (list of attributes)] (r_1)
And	&	$cond_1$ & $cond_2$ or $cond_1$ && $cond_2$
Or	|	$cond_1$ | $cond_2$ or $cond_1$ || $cond_2$
Not	!	$!cond_1$

Slika 2. Predložena sintaksa relacijske algebre (L. Malkova, 2009, 24. str.)

7.5 Stablo parsiranja

Kako smo ranije definirali sintaksu relacijske algebre, sada je potrebno definirati alat koji će omogućiti provjeru sintakse. U tu svrhu koristi se *parser* koji konstruira stablo parsiranja. Stablo parsiranja se kreira koristeći ranije spomenutu gramatiku *SQL* upitnog jezika. Mi smo naveli samo gramatiku *SELECT* naredbe, iako svaka od naredbi unutar *SQL-a* ima definiranu gramatiku jer u protivnom *SQL* kao upitni jezik ne bi bio ispravno definiran.

Kako bi razumjeli cijeli postupak kreiranja stabla parsiranja potrebno je definirati pojam semantičke akcije. Prilikom izvršavanja produkcijskog pravila, blok semantičke akcije kreira novi čvor u stablu parsiranja. Svaki čvor tog stabla sadrži informacije o nekoj od klauzula koje se nalaze u definiranom *SQL* upitu. Čvorovi koji nemaju djece se nazivaju listovima, a ti čvorovi su obično nazivi relacija te varijabli, konstante ili informacije koje se pojavljuju kao *subscript* operatora.

„Svako produkcijsko pravilo stvara djecu promatranog čvora u stablu parsiranja.“

(*Obrada upita* (prezentacija), 2019. , A. Lovrenčić)

Postupak kreiranja stabla parsiranja kreće od *non-terminal* (<*query*>), odnosno početnog simbola, te primjenjuje definirana produkcijska pravila sve dok ne dođe do terminal elementa.

„U predloženoj notaciji relacijske algebre izraz za kreiranje čvora bi imao sljedeći oblik :

```
exp 'INTERSECTION' exp {
        my %node;
        $node{type} = 'intersection';
        $node{left} = $_[1];
        $node{right} = $_[3];
        return \%node;
}“
```

(*Relational Algebra Expression Evaluation*, L. Malkova, 2009. , 26. str.)

Parser javlja grešku kada naiđe na izraz koji nije sintaktički ispravno definiran. Dio izraza koji *parser* nije uspio „prepoznati“ se vraća kao dio poruke pogreške, indirektno pokazujući gdje se dogodila greške te gdje se konkretno greška nalazi u definiranom izrazu.

7.5.1 Izrada stabla parsiranja

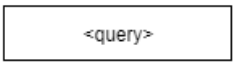
Kako bi mogli kreirati stablo parsiranja moramo imati definiran *SQL* upit. Ovakvim definiranjem upita podrazumijevamo da je izraz sintaktički ispravan i da se temeljem njega može kreirati odgovarajuće stablo parsiranja.

Primjer 31. Izrada stabla parsiranja

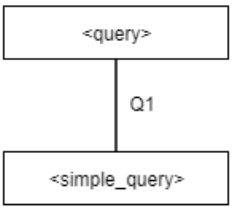
SQL(U) :

SELECT	Prezime, Ime
FROM	korisnici
WHERE	Ime
LIKE	'Marko'

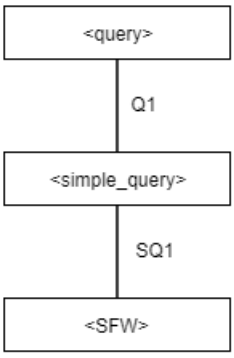
Koristeći definiranu gramatiku naredbe *SELECT* , *SQL* upit pretvaramo u stablo parsiranja.



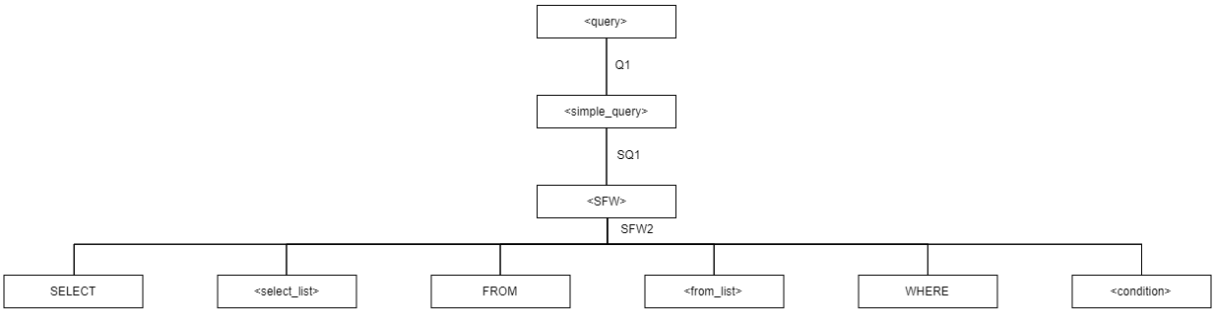
Q1 <query> ← <simple_query>;



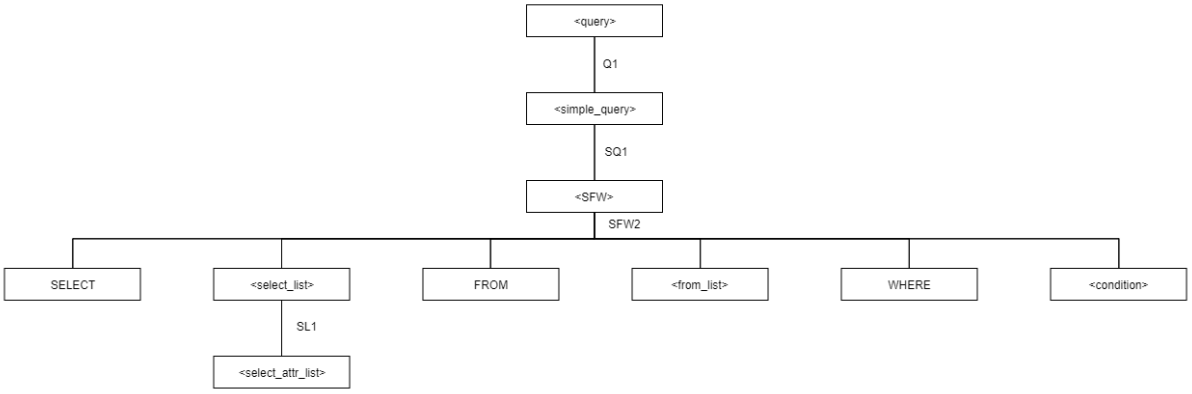
SQ1 <simple_query> ← <SFW>;



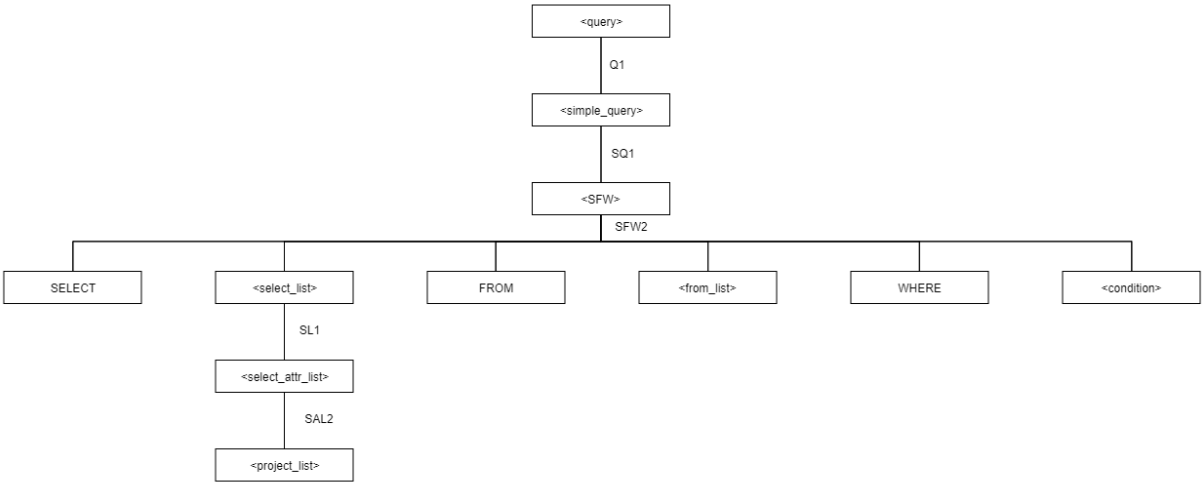
SFW2 <SFW> ← SELECT <select_list> FROM <from_list> WHERE <condition>;



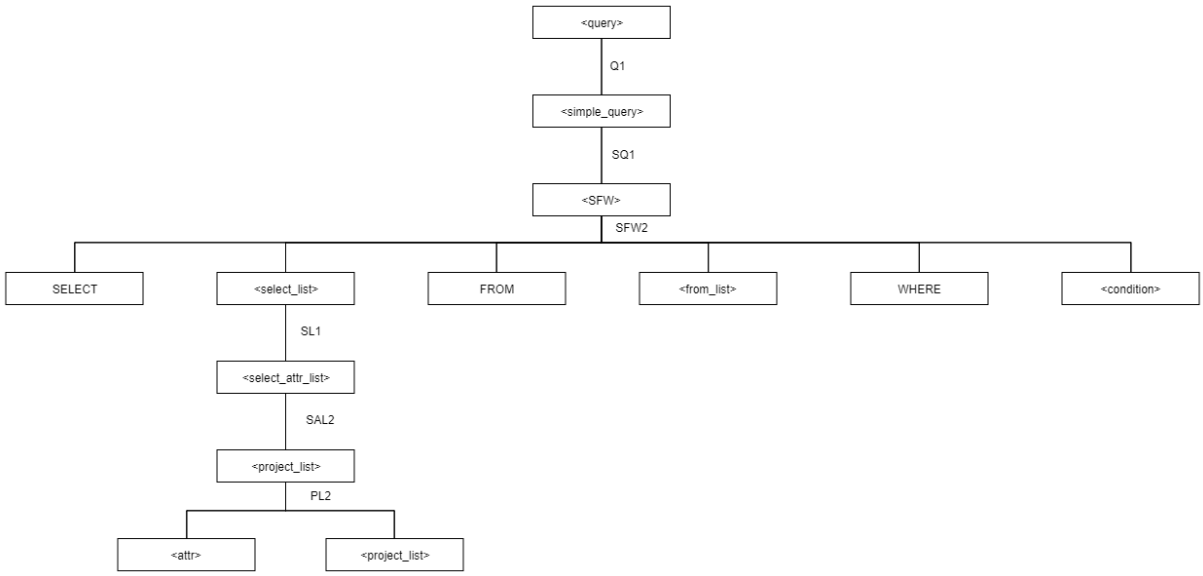
SL1 <select_list> ← <select_attr_list>;



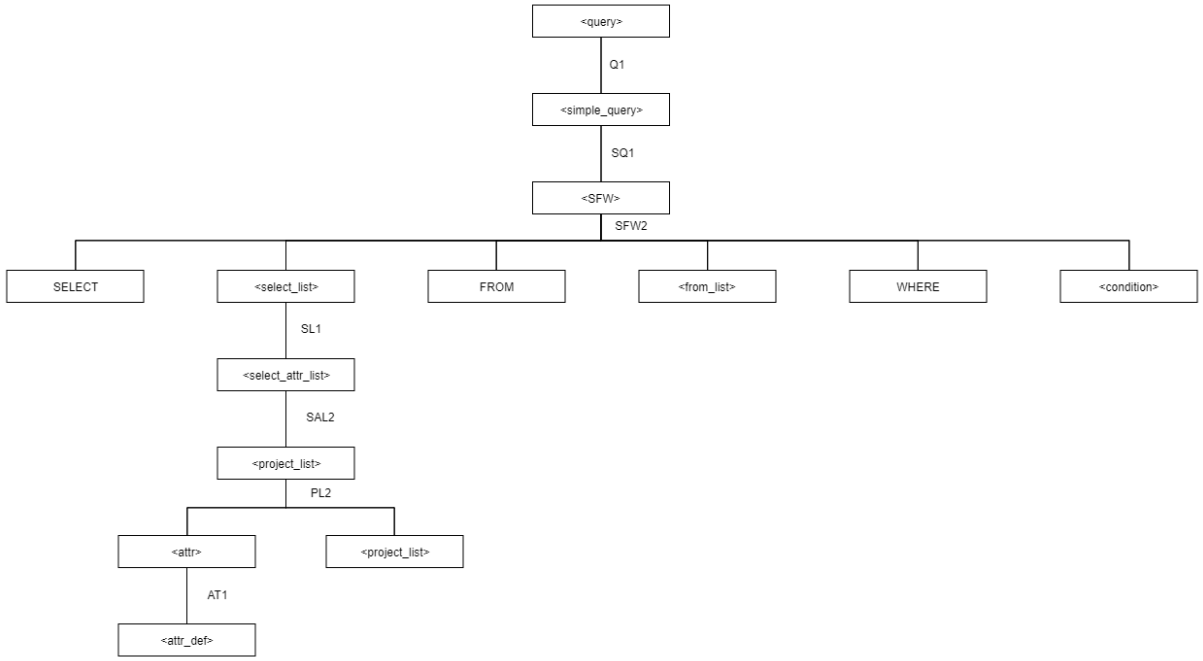
SAL2 <select_attr_list> ← <project_list>;



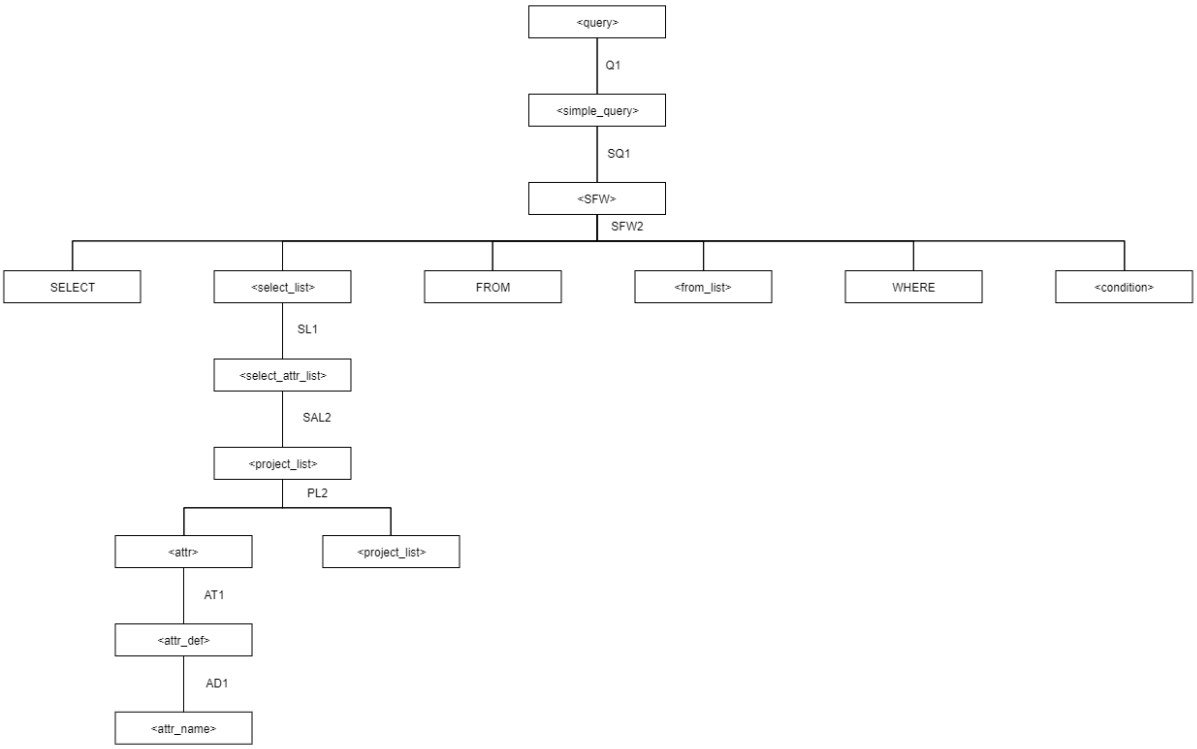
PL2 <project_list> ← <attr> , <project_list>;



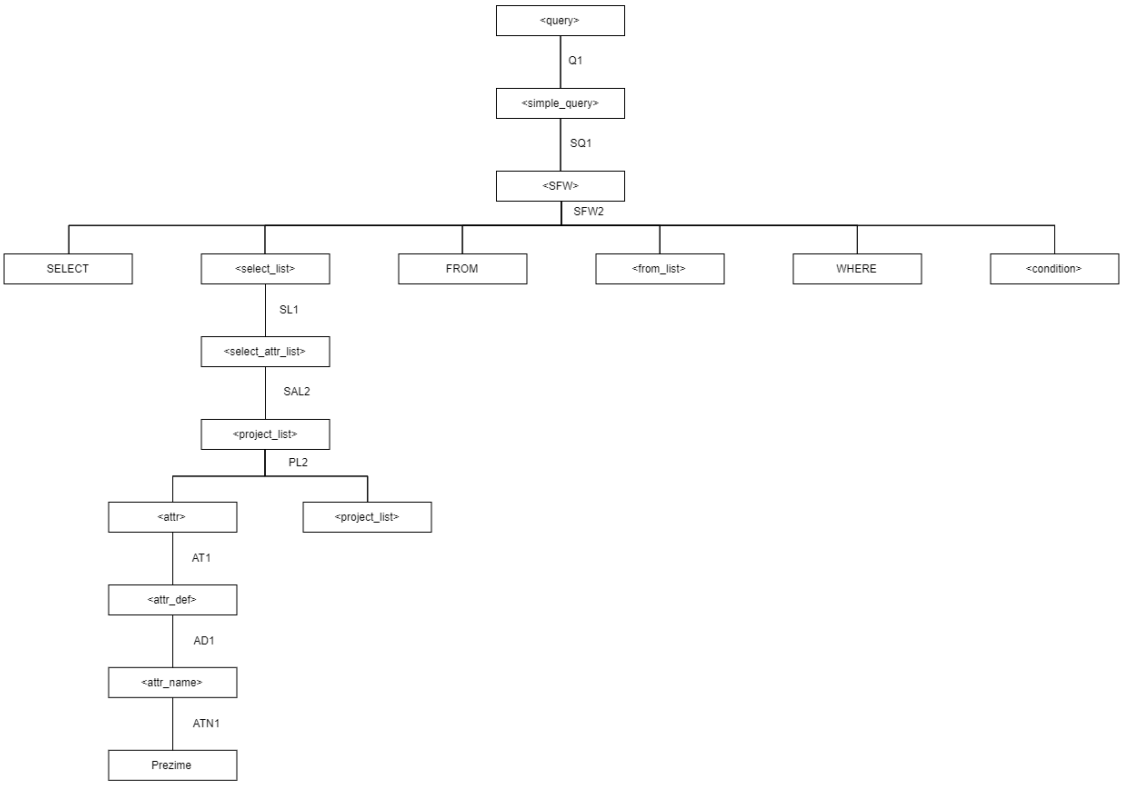
AT1 <attr> ← <attr_def>;



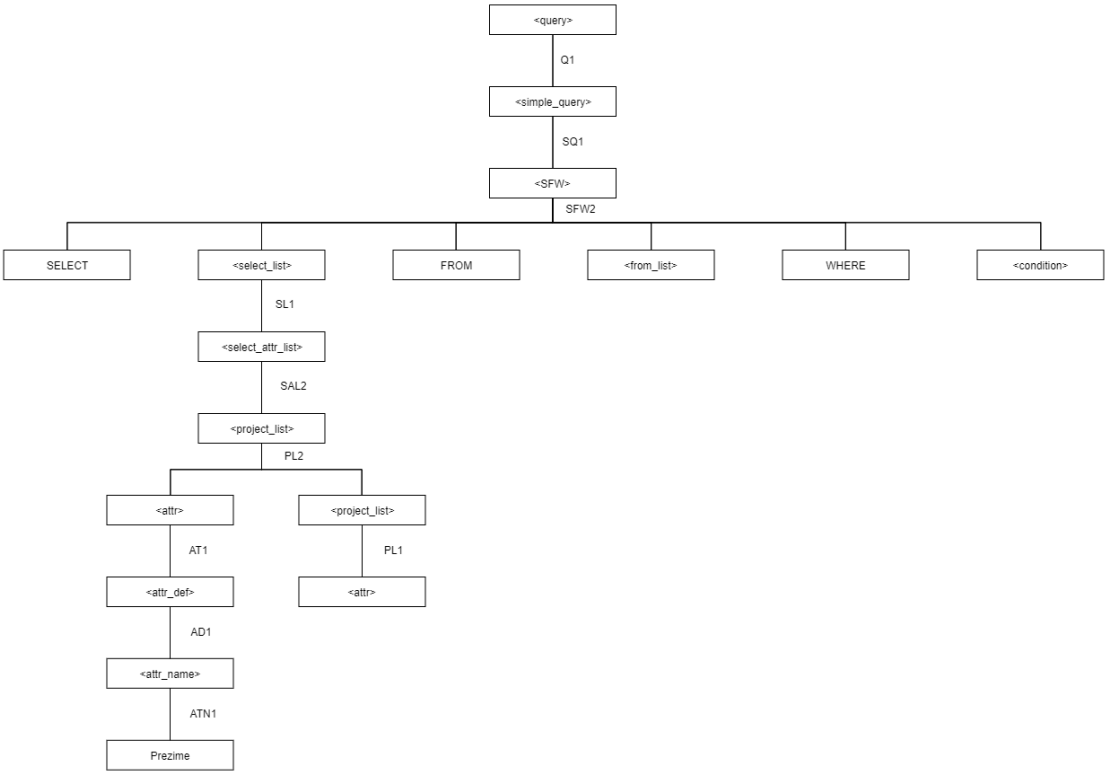
AD1 <attr_def> ← <attr_name>;



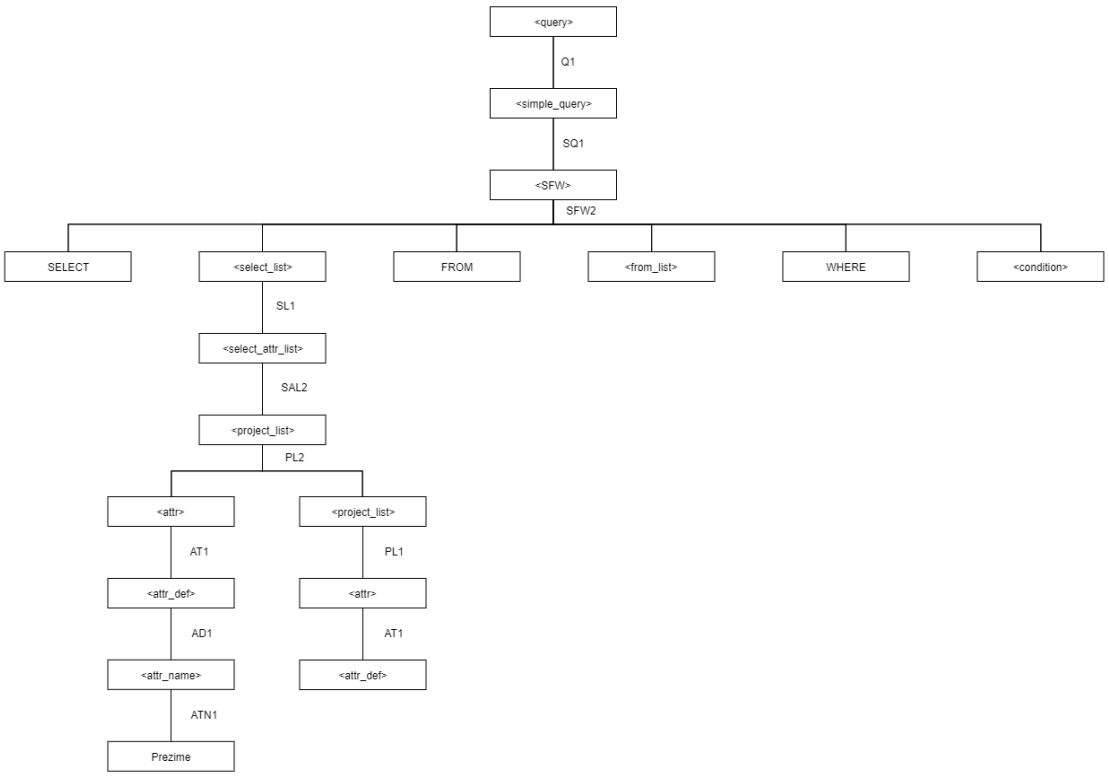
ATN1 <attr_name> ← Prezime;



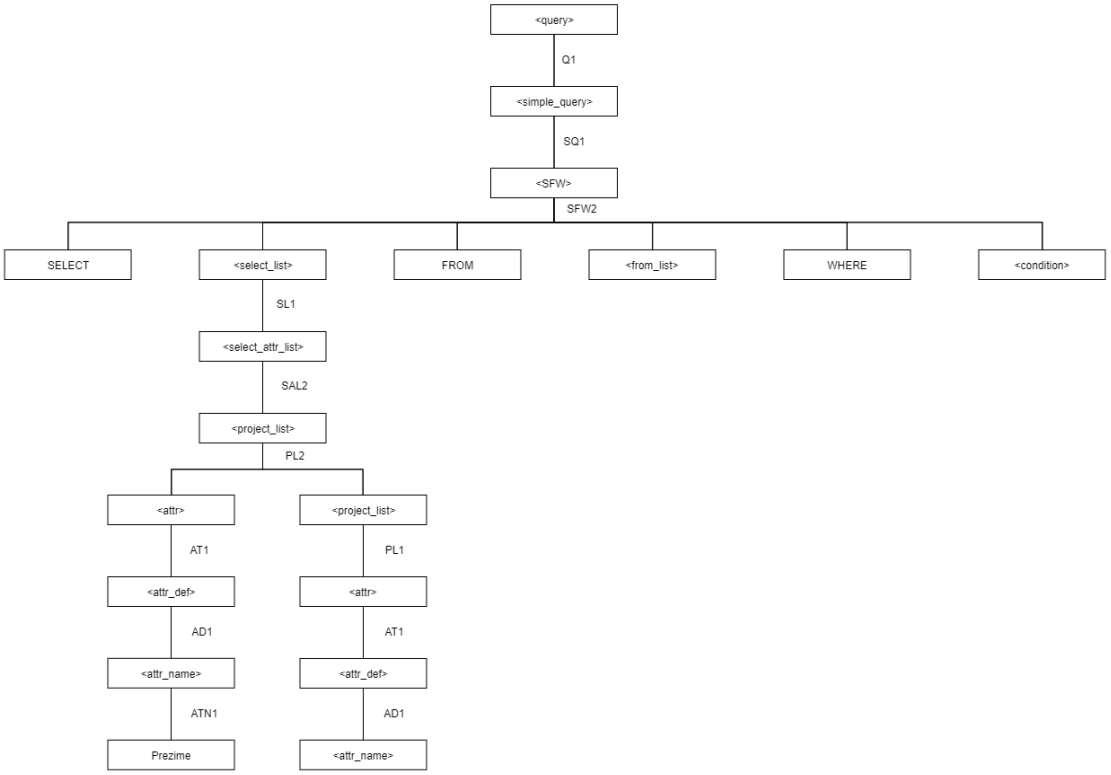
PL1 <project_list> ← <attr>;



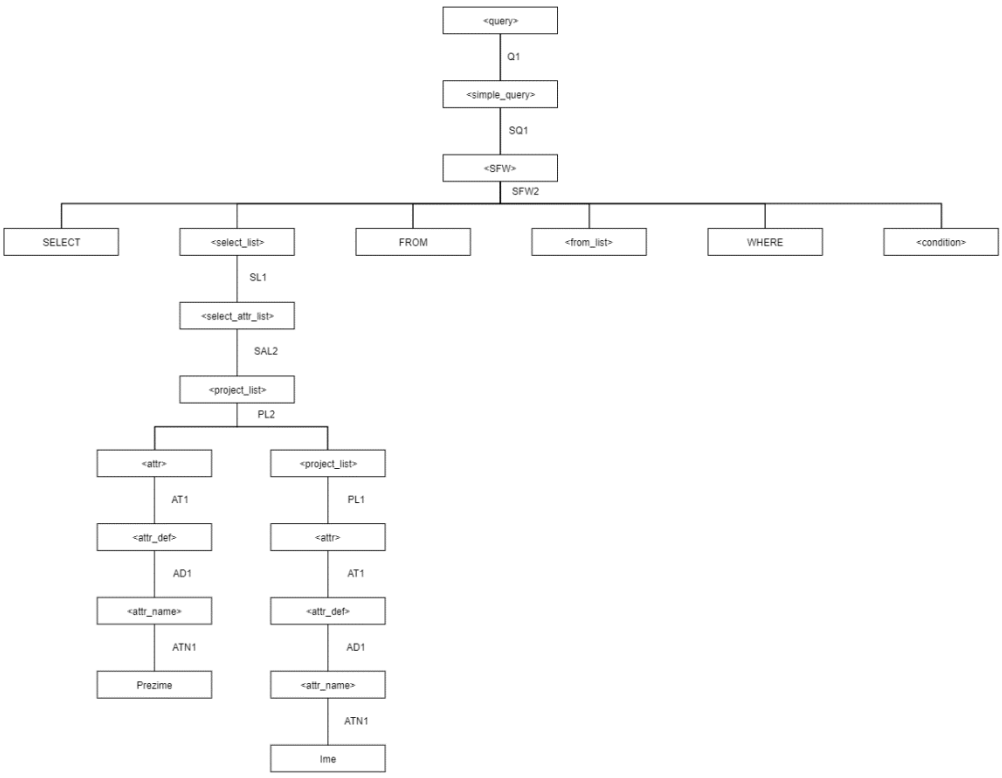
AT1 <attr> ← <attr_def>;



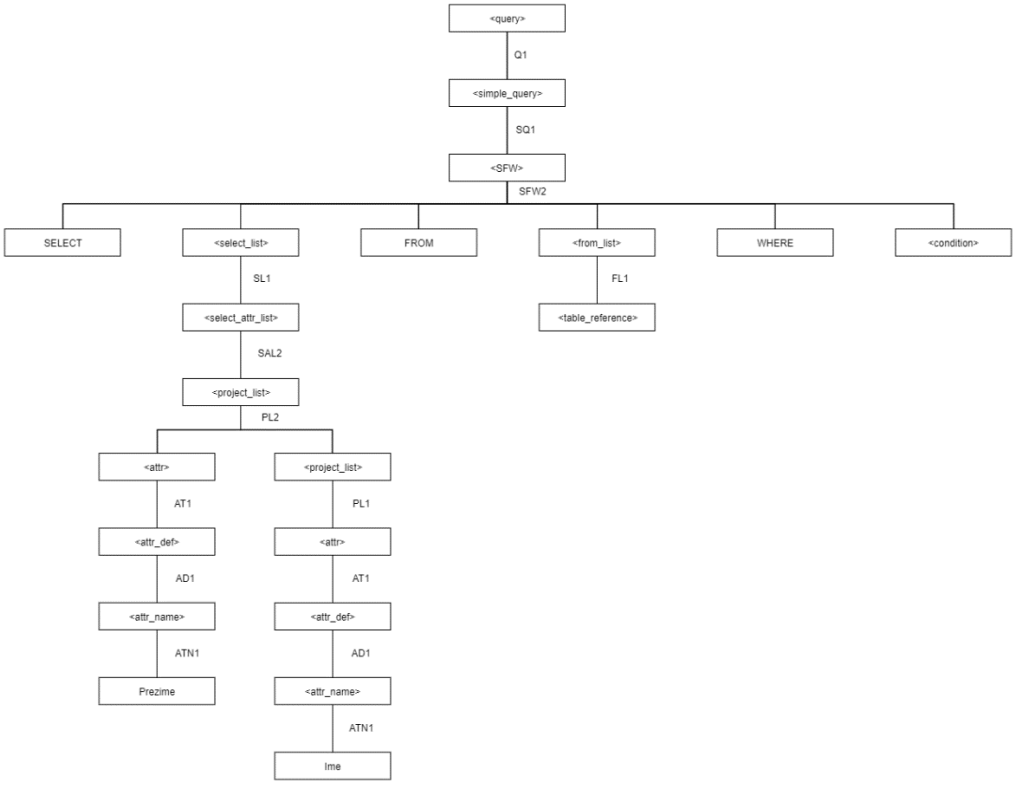
AD1 <attr_def> ← <attr_name>;



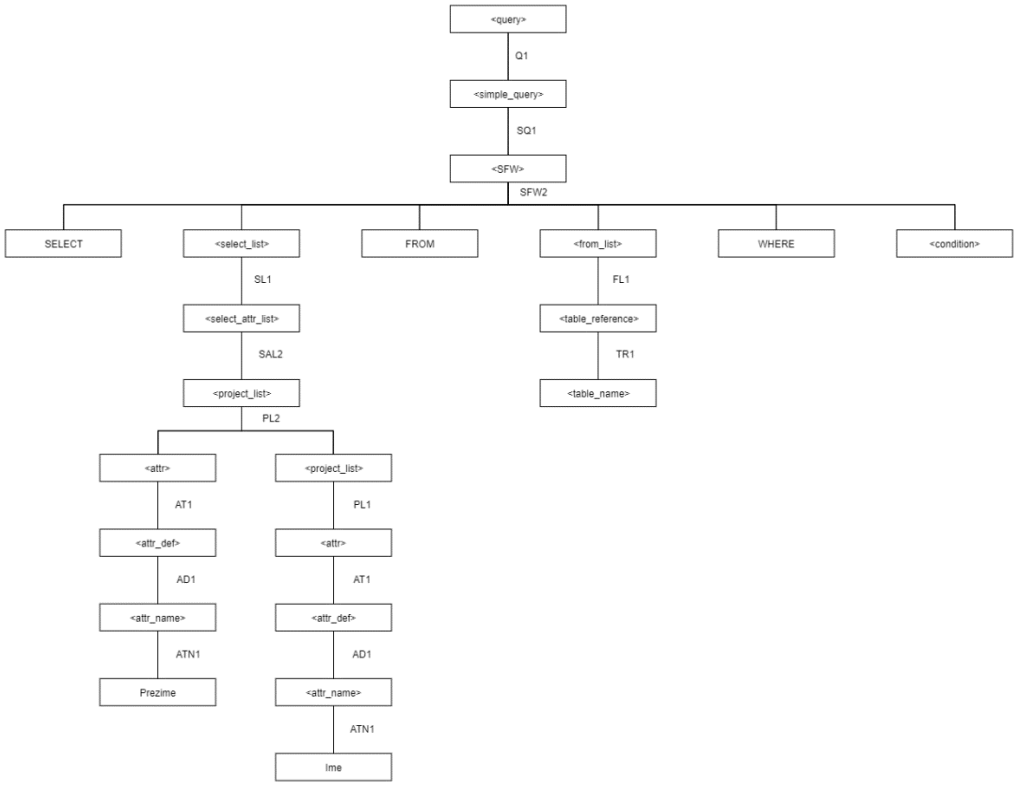
ATN1 <attr_name> ← Ime;



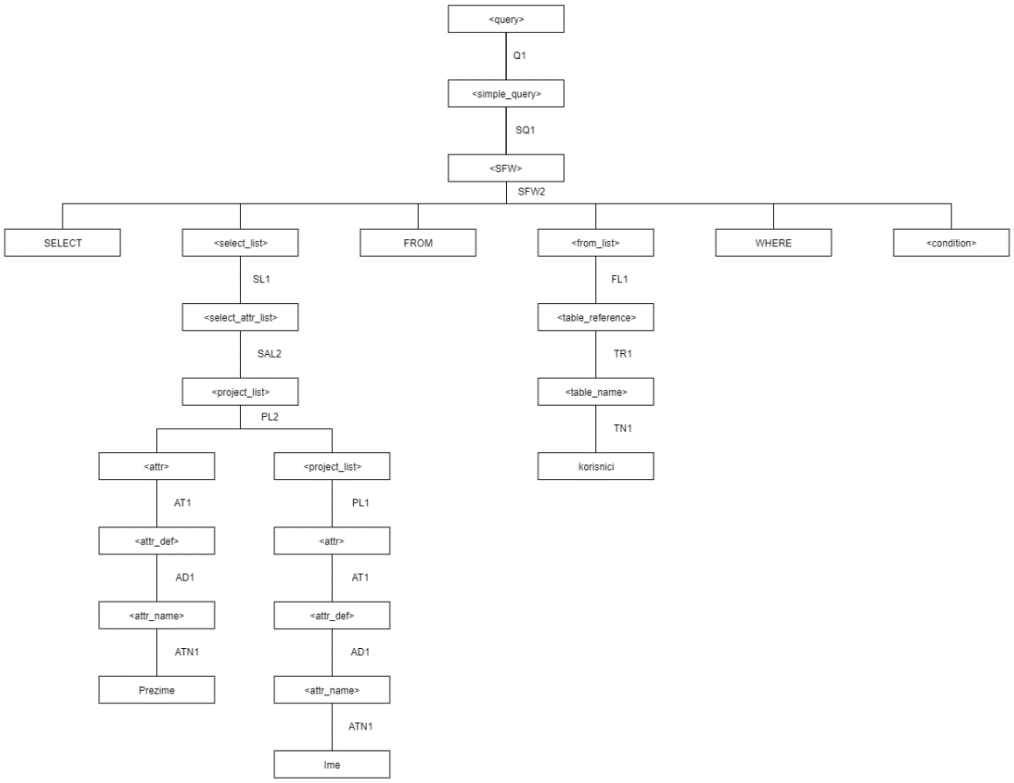
FL1 <from_list> ← <table_reference>;



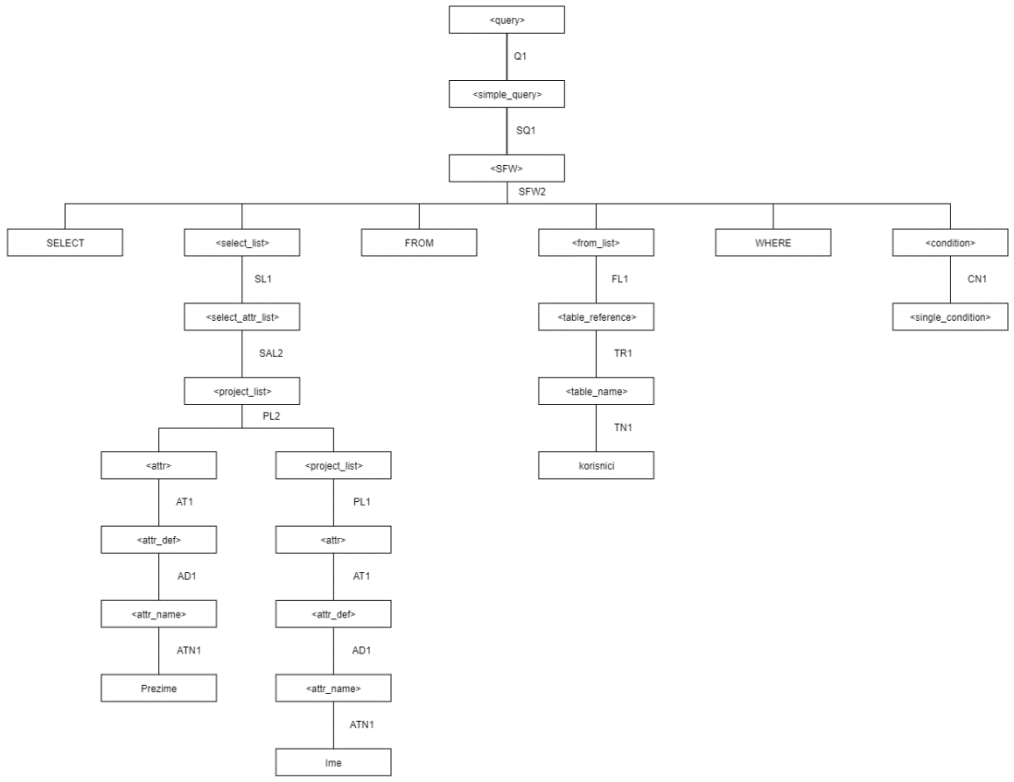
TR1 <table_reference> ← <table_name>;



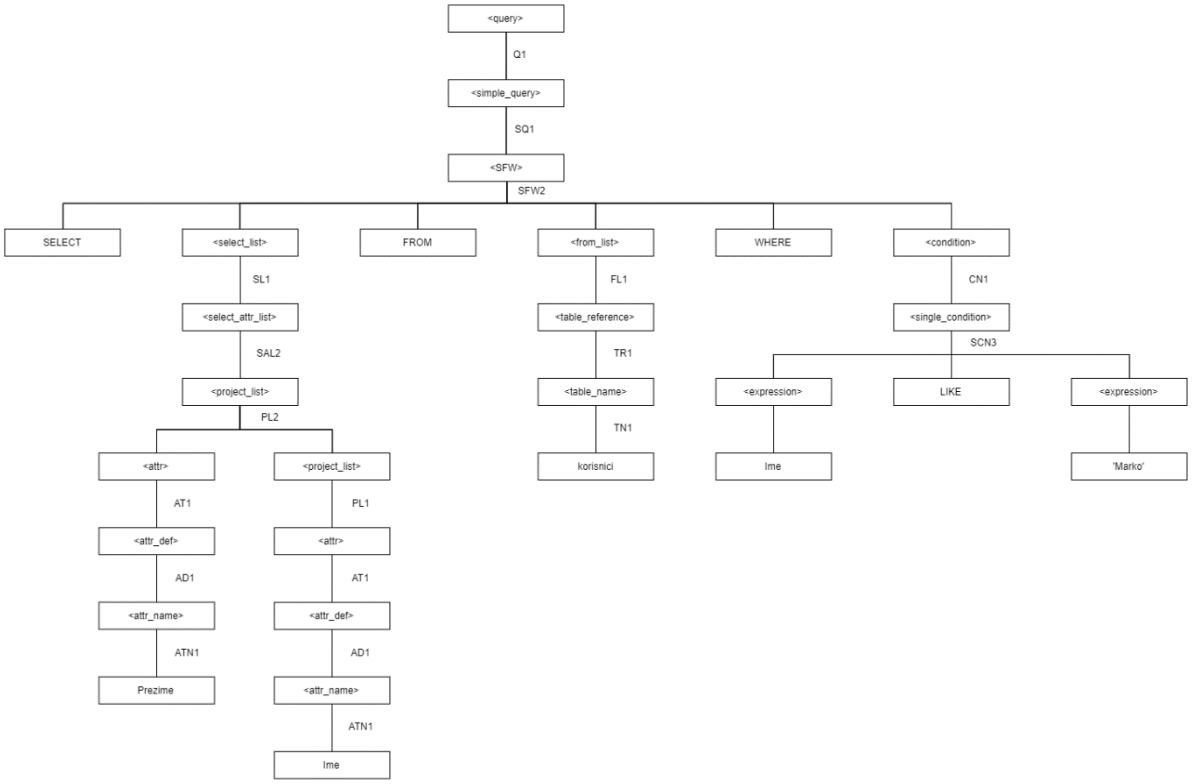
TN1 <table_name> ← korisnici;



CN1 <condition> ← <single_condition>;



SCN3 <single_condition> ← <expression> LIKE <expression>;



Ukoliko *parser* uspije kreirati stablo parsiranja tada indirektno provodi i sintaktičku analizu kojom se utvrđuje sintaktička ispravnost definiranog *SQL* upita. Na temelju kreiranog stabla parsiranja, *query preprocessor* komponenta, nakon provedene semantičke analize, kreira stablo algebarskih operatora koje reprezentira upit u relacijskoj algebri.
Pretpostavimo da je u definiranom upitu klauzula *LIKE* pogrešno napisana. Na taj način ćemo isforsirati sintaktičku grešku.

SQL(U) :

SELECT	Prezime, Ime
FROM	korisnici
WHERE	Ime
LIK	'Marko'

Parser u tom slučaju ne bi uspio kreirati stablo parsiranja te bi „izbacio“ ***Syntax Error***.

7.6 Semantička analiza

Neizostavan dio analize upitnog izraza je i semantička analiza. Semantička analiza provodi se u fazi pretprocesiranja upita. Kada smo govorili o *parseru*, kao komponenti koja je zadužena za kreiranje stabla parsiranja, tada smo napomenuli da se kreiranjem stabla parsiranja implicitno provodi sintaktička analiza.

Međutim, sintaktička analiza nije dovoljna kako bi mogli reći da je definirani upitni izraz korektan te je zbog toga potrebno provesti i semantičku analizu. Za provođenje semantičke analize zadužen je *query preprocessor*.

Semantičkom analizom se želi utvrditi da sve definirane relacije, kao i atributi tih relacija, u upitnom izrazu zaista i postoje u bazi podataka nad kojom će se izvršavati upit. Osim toga, potrebno je provjeriti kompatibilnost tipova podataka lijeve i desne strane u uvjetima selekcije kao i jesu li sve zagrade ispravno ugniježđene jedna unutar druge te korektnost primjene operacija na definirane operande.

„Semantička analiza je rekurzivna funkcija koja ophodi stablo parsiranja, počevši od listova, te provjerava postoji li relacija u bazi podataka."

(prema *Relational Algebra Expression Evaluation*, L. Malkova, 2009. , 27. str.)

Ukoliko se prilikom ophodnje stabla, odnosno analizom relacija, utvrdi da neka od relacija ne postoji u bazi podataka tada *query preprocessor* javlja grešku. Greška se obično manifestira kao poruka korisniku unutar *SUBP-a* u kojem radi.

SQL(U) :

SELECT	Prezime, Ime
FROM	korisnik
WHERE	Ime
LIKE	'Marko'

Semantic Error: Relation 'korisnik' does not exist.

Ukoliko analizirana relacija postoji tada se dohvaća relacijska shema relacije koja se zatim interno pohranjuje za daljnje korištenje. Prilikom ophodnje stabla, *query preprocessor* u svakom koraku zna koje relacije i atributi su dostupni za korištenje.

Nailaskom na određenu operaciju, provjerava se može li ista biti izvršena temeljem relacija i trenutno dostupnih atributa.

Primjer 32. Pretvorba *SQL* upita u upit relacijske algebre

Zadatak 2.20 – Zbrika zadataka iz baza podataka – Schatten M.

dimnjacar	#DimnjacarId	Prezime
	1	Mamić
	2	Huljić

ulica	#UlicaId	NazivUlice
	u_1	Jalkovečka ulica
	u_2	Ulica braće Slukan
	u_3	Ulica braće Radić

zaduzenje	Dimnjacar	Ulica
	1	u_1
	2	u_3
	1	u_2

Neka je zadan upit *U :*
Pronađi sve nazive ulica za koje je zadužen dimnjačar s prezimenom Mamić.

Koraci pretvorbe *SQL* upita u upit relacijske algebre :

1. Kako bi mogli napraviti konverziju iz *SQL* upita u upit relacijske algebre, prvo je potrebno definirati odgovarajući *SQL* upit.

SQL(U) :

```
SELECT          NazivUlice
FROM            dimnjacar
INNER JOIN      zaduzenje
ON              DimnjacarId = Dimnjacar
INNER JOIN      ulica
ON              UlicaId = Ulica
WHERE           Prezime = 'Mamić'
```

Dakle, potrebno je dohvatiti sve nazive ulica za koje je zadužen dimnjačar s prezimenom *Mamić*. Nazivi ulica definirani su atributom *NazivUlice* u relaciji *ulica*, dok su prezimena dimnjačara definirana atributom *Prezime* u relaciji *dimnjacar*. Kako svaki dimnjačar može biti zadužen za više ulica, u prikazanom modelu pojavljuje se relacija, slabi entitet *zaduzenje*. Svaki zapis te relacija se referencira na *dimnjacara* te *ulicu* za koju je taj dimnjačar zadužen. Kako bi došli do traženog naziva ulice potrebno je povezati relacije *ulica*, *zaduzenje* i *dimnjacar* primjenom unutarnjeg spoja (*INNER JOIN*) te zatim filtrirati dobivene slogove rezultata povezivanja prema prezimenu *Mamić.*

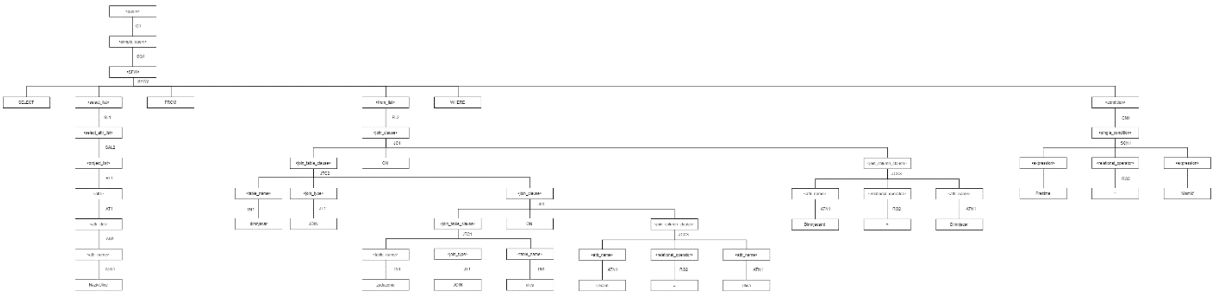
Uz poznavanje osnova baza podataka, jednostavno je odrediti slogove koje će kao rezultat vratiti postavljeni *SQL* upit.

◦U	NazivUlice
	Jalkovečka ulica
	Ulica braće Slukan

Ono što se dešava u pozadini baze podataka prilikom izračuna rezultata za postavljeni upit je izvođenje operacija relacijske algebre. Cijeli postupak izvođenja operacija relacijske algebre bit će prikazan u nastavku.

2. Sintaktička analiza i kreiranje stabla parsiranja

Parser komponenta na temelju postavljenog *SQL* upita pokušava kreirati stablo parsiranja. Ukoliko je stablo parsiranja uspješno kreirano tada sa sigurnošću možemo reći da je definirani upit i sintaktički ispravan. Nakon sintaktičke analiza provodi se semantička analiza te ukoliko je upit i semantički ispravan tada *query optimizer* generira upitne planove te između njih odabire onaj koji je vremenski optimalan za izvršavanje.



Slika 3. Stablo parsiranja Primjer 32.

3. Pretvorba *SQL* upita u upit relacijske algebre

Postupak prevođenja u upit relacijske algebre obavlja *query preprocessor* komponenta na temelju kreiranog stabla parsiranja iz prethodnog koraka.

$$\boldsymbol{RA(U)}: \Pi_{\text{NazivUlice}}(\sigma_{F}(\text{dimnjacar} \otimes \text{zaduzenje} \otimes \text{ulica}))$$

$$F : (\text{prezime} = \text{'Mamić'}) \wedge (\text{DimnjacarId} = \text{Dimnjacar}) \wedge (\text{UlicaId} = \text{Ulica})$$

Sada ćemo na temelju kreiranog upita relacijske algebre izračunati rezultat upita. Izračunavanje odgovora na upit bit će prikazano korak po korak kao što se ono provodi tijekom izvršavanja upita u *SUBP-u*.
Kao što je ranije rečeno, prvo je potrebno izračunati unutarnje operacije kako bi rezultat tih operacije mogle koristiti druge operacije.

4. Izvršavanje upita relacijske algebre

s_1 = dimnjacar ⊗ zaduzenje ⊗ ulica

Zbog jednostavnosti zapisa koristit ćemo skraćeni zapis, odnosno pokrate naziva ulica.

Ju – Jalkovečka ulica

UbS – Ulica braće Slukan

UbR – Ulica braće Radića

s_1	#DimnjacarId	Prezime	Dimnjacar	Ulica	#UlicaId	NazivUlice
t_1	1	Mamić	1	u_1	u_1	Ju
t_2	1	Mamić	1	u_1	u_2	UbS
t_3	1	Mamić	1	u_1	u_3	UbR
t_4	1	Mamić	2	u_3	u_1	Ju
t_5	1	Mamić	2	u_3	u_2	UbS
t_6	1	Mamić	2	u_3	u_3	UbR
t_7	1	Mamić	1	u_2	u_1	Ju
t_8	1	Mamić	1	u_2	u_2	UbS
t_9	1	Mamić	1	u_2	u_3	UbR
t_{10}	2	Huljić	1	u_1	u_1	Ju
t_{11}	2	Huljić	1	u_1	u_2	UbS
t_{12}	2	Huljić	1	u_1	u_3	UbR
t_{13}	2	Huljić	2	u_3	u_1	Ju
t_{14}	2	Huljić	2	u_3	u_2	UbS
t_{15}	2	Huljić	2	u_3	u_3	UbR
t_{16}	2	Huljić	1	u_2	u_1	Ju
t_{17}	2	Huljić	1	u_2	u_2	UbS
t_{18}	2	Huljić	1	u_2	u_3	UbR

$s_2 = \sigma_F(\text{dimnjacar} \otimes \text{zaduzenje} \otimes \text{ulica}) \equiv \sigma_F(s_1)$

F : (prezime = 'Mamić') ∧ (DimnjacarId = Dimnjacar) ∧ (UlicaId = Ulica)

s_2	#DimnjacarId	Prezime	Dimnjacar	Ulica	#UlicaId	NazivUlice
t_1	1	Mamić	1	u1	u1	Ju
t_8	1	Mamić	1	u2	u2	UbS

t_1: (Mamić = Mamić) ∧ (1 = 1) ∧ (u1 = u1) ≡ 1 ∧ 1 ∧ 1 ≡ 1

t_2: (Mamić = Mamić) ∧ (1 = 1) ∧ (u2 = u1) ≡ 1 ∧ 1 ∧ 0 ≡ 0

t_3: (Mamić = Mamić) ∧ (1 = 1) ∧ (u3 = u1) ≡ 1 ∧ 1 ∧ 0 ≡ 0

t_4: (Mamić = Mamić) ∧ (1 = 2) ∧ (u1 = u3) ≡ 1 ∧ 0 ∧ 0 ≡ 0

t_5: (Mamić = Mamić) ∧ (1 = 2) ∧ (u2 = u3) ≡ 1 ∧ 0 ∧ 0 ≡ 0

t_6: (Mamić = Mamić) ∧ (1 = 2) ∧ (u3 = u3) ≡ 1 ∧ 0 ∧ 1 ≡ 0

t_7: (Mamić = Mamić) ∧ (1 = 1) ∧ (u1 = u2) ≡ 1 ∧ 1 ∧ 0 ≡ 0

t_8: (Mamić = Mamić) ∧ (1 = 1) ∧ (u2 = u2) ≡ 1 ∧ 1 ∧ 1 ≡ 1

t_9: (Mamić = Mamić) ∧ (1 = 1) ∧ (u3 = u2) ≡ 1 ∧ 0 ∧ 1 ≡ 0

t_{10}: (Huljić = Mamić) ∧ (2 = 1) ∧ (u1 = u1) ≡ 0 ∧ 0 ∧ 1 ≡ 0

t_{11}: (Huljić = Mamić) ∧ (2 = 1) ∧ (u2 = u1) ≡ 0 ∧ 0 ∧ 0 ≡ 0

t_{12}: (Huljić = Mamić) ∧ (2 = 1) ∧ (u3 = u1) ≡ 0 ∧ 0 ∧ 0 ≡ 0

t_{13}: (Huljić = Mamić) ∧ (2 = 2) ∧ (u1 = u3) ≡ 0 ∧ 1 ∧ 0 ≡ 0

t_{14}: (Huljić = Mamić) ∧ (2 = 2) ∧ (u2 = u3) ≡ 0 ∧ 1 ∧ 0 ≡ 0

t_{15}: (Huljić = Mamić) ∧ (2 = 2) ∧ (u3 = u3) ≡ 0 ∧ 1 ∧ 1 ≡ 0

t_{16}: (Huljić = Mamić) ∧ (2 = 1) ∧ (u1 = u2) ≡ 0 ∧ 0 ∧ 0 ≡ 0

t_{17}: (Huljić = Mamić) ∧ (2 = 1) ∧ (u2 = u2) ≡ 0 ∧ 0 ∧ 1 ≡ 0

t_{18}: (Huljić = Mamić) ∧ (2 = 1) ∧ (u3 = u2) ≡ 0 ∧ 0 ∧ 0 ≡ 0

$s_3 = \Pi_{\text{NazivUlice}}(\sigma_F(\text{dimnjacar} \otimes \text{zaduzenje} \otimes \text{ulica})) \equiv \Pi_{\text{NazivUlice}}(s_2)$

◦U = s_3	NazivUlice
	Jalkovečka ulica
	Ulica braće Slukan

Primjer 33. Pretvorba *SQL* upita u upit relacijske algebre

Zadatak 2.16 – Zbrika zadataka iz baza podataka – Schatten M.

vlasnik	#VlasnikId	Prezime
	1	Lukšić
	2	Badanjak

agent	#AgentId	Prezime
	1	Anić
	2	Ivić

nekretnina	#NekretninaId	Adresa	Agent	Vlasnik
	n_1	Jalkovečka 21	1	2
	n_2	Dugopoljska 3	2	1
	n_3	Butorčeva 17	1	2

Neka je zadan upit *U :*
Pronađi sve adrese nekretnina, prezimena i imena agenata koji su dodijeljeni toj nekretnini te prezimena vlasnika koji prodaju zadanu nekretninu.

Koraci pretvorbe *SQL* upita u upit relacijske algebre :

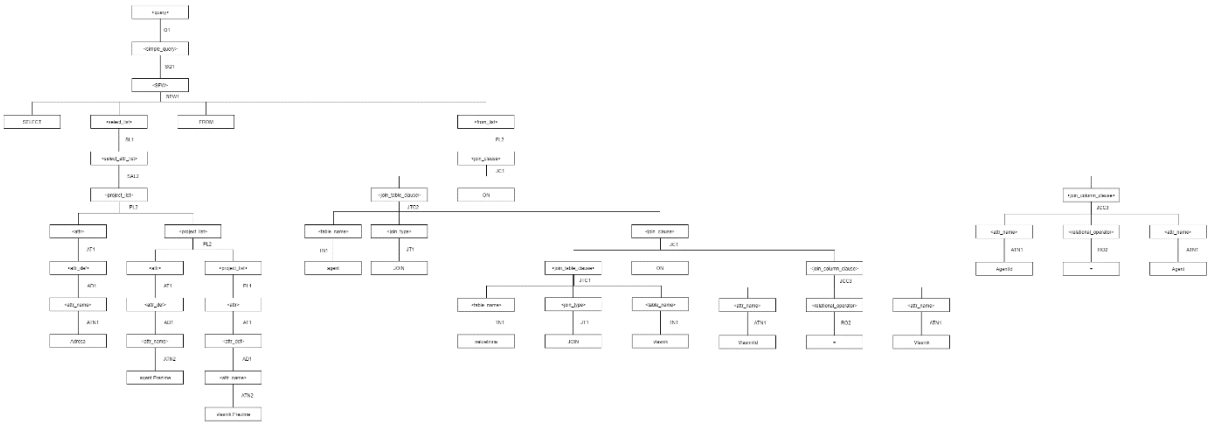
1. Definiranje *SQL* upita

SQL(U) :

SELECT	Adresa, agent.Prezime, vlasnik.Prezime
FROM	agent
INNER JOIN	nekretnina
ON	AgentId = Agent
INNER JOIN	vlasnik
ON	VlasnikId = Vlasnik

2. Sintaktička analiza i kreiranje stabla parsiranja

Parser komponenta na temelju postavljenog *SQL* upita pokušava kreirati stablo parsiranja. Ukoliko je stablo parsiranja uspješno kreirano tada možemo sa sigurnošću reći da je definirani upit i sintaktički ispravan. Nakon sintaktičke analiza provodi se semantička analiza te ukoliko je upit i semantički ispravan tada *query optimizer* generira upitne planove te između njih odabire onaj koji je vremenski optimalan za izvršavanje.



Slika 4. Stablo parsiranja Primjer 33.

3. Pretvorba *SQL* upita u upit relacijske algebre

Postupak prevođenja u upit relacijske algebre obavlja *query preprocessor* komponenta na temelju kreiranog stabla parsiranja iz prethodnog koraka.

$$\textbf{\textit{RA(U)}} : \Pi_{\text{Adresa, agent.Prezime, vlasnik.Prezime}} \left(\sigma_{F} (\text{agent} \otimes \text{nekretnina} \otimes \text{vlasnik})\right)$$

$$F : (\text{AgentId} = \text{Agent}) \wedge (\text{VlasnikId} = \text{Vlasnik})$$

Sada ćemo na temelju kreiranog upita relacijske algebre izračunati rezultat upita. Izračunavanje odgovora na upit bit će prikazano korak po korak kao što se ono provodi tijekom izvršavanja upita u *SUBP-u*.

Kao što je ranije rečeno, prvo je potrebno izračunati unutarnje operacije kako bi rezultat tih operacije mogle koristiti druge operacije.

4. Izvršavanje upita relacijske algebre

s_1 = agent ⊗ nekretnina ⊗ vlasnik

Zbog jednostavnosti zapisa koristit ćemo skraćeni zapis, odnosno pokrate adresa.

J21 – Jalkovečka 21

D3 – Dugopoljska 3

B17– Butorčeva17

s_1	#AgentId	a.Prezime	#NekretninaId	Adresa	Agent	Vlasnik	VlasnikId	v.Prezime
t_1	1	Anić	n_1	J21	1	2	1	Lukšić
t_2	1	Anić	n_1	J21	1	2	2	Badanjak
t_3	1	Anić	n_2	D3	2	1	1	Lukšić
t_4	1	Anić	n_2	D3	2	1	2	Badanjak
t_5	1	Anić	n_3	B17	1	2	1	Lukšić
t_6	1	Anić	n_3	B17	1	2	2	Badanjak
t_7	2	Ivić	n_1	J21	1	2	1	Lukšić
t_8	2	Ivić	n_1	J21	1	2	2	Badanjak
t_9	2	Ivić	n_2	D3	2	1	1	Lukšić
t_{10}	2	Ivić	n_2	D3	2	1	2	Badanjak
t_{11}	2	Ivić	n_3	B17	1	2	1	Lukšić
t_{12}	2	Ivić	n_3	B17	1	2	2	Badanjak

$s_2 = \sigma_F(\text{agent} \otimes \text{nekretnina} \otimes \text{vlasnik}) \equiv \sigma_F(s_1)$

F : (AgentId = Agent) ∧ (VlasnikId = Vlasnik)

s_1	#AgentId	a.Prezime	#NekretninaId	Adresa	Agent	Vlasnik	VlasnikId	v.Prezime
t_2	1	Anić	n_1	J21	1	2	2	Badanjak
t_6	1	Anić	n_3	B17	1	2	2	Badanjak
t_9	2	Ivić	n_2	D3	2	1	1	Lukšić

t_1: (1 = 1) ∧ (1 = 2) ≡ 1 ∧ 0 ≡ 0

t_2: (1 = 1) ∧ (2 = 2) ≡ 1 ∧ 1 ≡ 1

t_3: (1 = 2) ∧ (1 = 1) ≡ 0 ∧ 1 ≡ 0

t_4: (1 = 2) ∧ (2 = 1) ≡ 0 ∧ 0 ≡ 0

t_5: (1 = 1) ∧ (1 = 2) ≡ 1 ∧ 0 ≡ 0

t_6: (1 = 1) ∧ (2 = 2) ≡ 1 ∧ 1 ≡ 1

t_7: (2 = 1) ∧ (1 = 2) ≡ 0 ∧ 0 ≡ 0

t_8: (2 = 1) ∧ (2 = 2) ≡ 0 ∧ 1 ≡ 0

t_9: (2 = 2) ∧ (1 = 1) ≡ 1 ∧ 1 ≡ 1

t_{10}: (2 = 2) ∧ (2 = 1) ≡ 1 ∧ 0 ≡ 0

t_{11}: (2 = 1) ∧ (1 = 2) ≡ 0 ∧ 0 ≡ 0

t_{12}: (2 = 1) ∧ (2 = 2) ≡ 0 ∧ 1 ≡ 0

$s_3 = \Pi_{\text{Adresa, agent.Prezime, vlasnik.Prezime}}(\sigma_F(\text{agent} \otimes \text{nekretnina} \otimes \text{vlasnik})) \equiv$
$\equiv \Pi_{\text{Adresa, agent.Prezime, vlasnik.Prezime}}(s_2)$

◦U = s_3	Adresa	a.Prezime	v.Prezime
t_2	J21	Anić	Badanjak
t_6	B17	Anić	Badanjak
t_9	D3	Ivić	Lukšić

8. Optimizacija upita

Relacijski upitni jezici, odnosno jezici za relacijske baze podataka ne postavljaju stroga ograničenja u postavljanju upita. Korisniku se daje velika sloboda kod postavljanja upita što rezultira upitima različite razine složenosti za dohvaćanje istog skupa podataka. „Teret efikasnog odgovaranja na te raznolike upite prebačen je na *DBMS (Database Management System)*, odnosno *SUBP*.“ (*Baze podataka skripta*, R. Manger, 2003. , 31. str.)

Razlog tome je što se odgovor na upit najčešće može dobiti na različite način, a zadatak *DBMS-a* je da odabere najefikasniji način kojim će vrijeme izvršavanja upita reducirati na minimum. U novije vrijeme, zbog ubrzanog razvoja tehnologije, potreba za uštedom podatkovnog prostora se zanemaruje te se velika pažnja posvećuje optimizaciji vremenske komponente. U ranijim primjerima, prilikom pretvorba iz *SQL-a* u upit relacijske algebre, vidljivo je da su upiti relacijske algebre vrlo složeni, a samim time i zahtjevni za izvršavanje.

Nakon prevođenja upita želi se postići optimalno vrijeme njegovog izvršavanja zbog čega je kreirani upitni plan potrebno optimizirati.

Odabir najefikasnijeg načina za odgovaranje na upit zove se optimizacija upita.

Za upit kažemo da je optimalan ukoliko se ne može primijeniti niti jedno od pravila optimizacije, odnosno ako upit zadovoljava pravilo minimalnosti.

„Današnji sustavi za upravljanje bazama podataka provode optimizaciju na dvije razine :

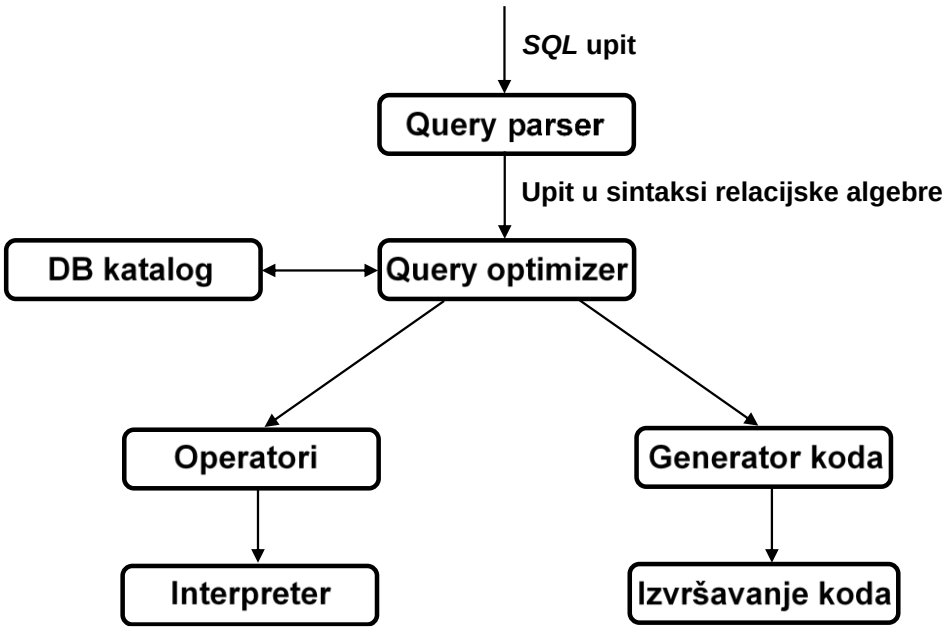
1. **viša (logička) razina** – reformuliranje algebarskog izraza u oblik koji je ekvivalentan polaznom, ali je pogodniji sa stanovišta izvrednjavanja
2. **niža (fizička) razina** – odabir efikasnog algoritma za izvrednjavanje svake od osnovnih operacija u algebarskom izrazu. Pritom se nastoji iskoristiti prisustvo pomoćnih struktura podataka, kao što su indeksi, kojima se direktno ubrzava pretraživanje podataka. Međutim, indeksi kao pomoćna struktura imaju i svojih nedostataka koji se očituju kod ažuriranja podataka jer osim samih podataka potrebno je ažurirati i indekse kako ne bi došlo do podatkovne neusklađenosti.“

(prema *Baze podataka skripta*, R. Manger, 2003. , 31. str.)

Optimizaciju upita obavlja *optimizer* komponenta koja je zadužena za odabir najefikasnijeg upitnog plana. *Optimizer* generira alternativne upitne planove te odabire onaj čija je cijena izvršavanja najmanja. Osim toga, *optimizer* komponenta treba prevesti odabrani plan u izvršni program koji se zatim izvršava.

8.1 Query optimizer komponenta

Query optimizer komponenta je zadužena za generiranje alternativnih upitnih planova te odabir najefikasnijeg upitnog plana. Nakon odabira najefikasnijeg upitnog plana, *optimizer* komponenta treba prevesti odabrani plan u izvršni program koji se zatim izvršava te nakon izvršavanja vraća kao odgovor na postavljeni upit.



Slika 5. Query optimizer

Prilikom optimizacije upita, *optimizer* komponenta se oslanja na katalog baze podataka koji sadrži metapodatke (imena relacija, relacijske sheme, domene podataka) te statistiku svake od relacija (broj slogova, broj atributa). Prilikom optimizacije razlikujemo dva slijeda koje *optimizer* komponenta može slijediti. Prvi slijed je standardni slijed koji se u današnje vrijeme primjenjuje u većini baza podataka. U tom slijedu *optimizer* se oslanja na tzv. predkompilirane module koji se u terminima baza podataka često nazivaju i operatorima. Operatore možemo poistovjetiti s pozivima funkcija neke biblioteke. O*ptimizer* nakon toga gradi stablo u čijim čvorovima se nalaze upravo ti operatori. Generirano stablo se naziva upitnim planom. Odabirom optimalnog slijeda izvršavanja operacija dolazimo do optimalnog plana kojeg je zatim potrebno interpretirati. Interpretacija se obavlja na način da *optimizer*, prilikom prolaska kroz stablo optimalnog upitnog plana, u svakom čvoru pozove funkciju operatora tog čvora. Nakon interpretacije upitnog plana kreira se izvršni program nakon čijeg izvršavanja dobivamo odgovor na postavljeni upit. Drugi slijed se rjeđe koristi pa ga iz tog razloga nećemo detaljnije razlagati.

8.2 Pravila za optimizaciju upita

Značajna ušteda vremena prilikom izračunavanja rezultata upita postiže se promjenom redoslijeda izvršavanja operacija s ciljem da se što prije smanji veličina relacija s kojima radimo. Smanjivanjem relacija, odnosno reduciranjem broja slogova relacije, postiže se da neka operacija pretražuje manji skup podataka koji dovodi do istog rješenja što u konačnici rezultira efikasnijim načinom rada sustava u cjelini.

Osnovna pravila optimizacije upita :

1. kombiniranje selekcija
2. izvlačenje selekcije ispred spoja ili produkta
3. izvlačenje selekcije ispred projekcije
4. kombiniranje projekcija
5. izvlačenje projekcije ispred spoja
6. optimizacija skupovnih operacija
7. pretvorba Kartezijevog produkta u spoj
8. pravila vezana uz selekciju
9. sortiranje i grupiranje izvršiti što je kasnije moguće

8.2.1 Kombiniranje selekcija

$$\sigma_{F2}(\sigma_{F1}(r)) \equiv \sigma_{F}(r),$$

gdje je $F : F_1 \wedge F_2$

Kombiniranjem selekcija smanjuje se potrebno vrijeme za izvršavanje upita ukoliko se obje selekcije odvijaju podjednako sporo, odnosno ako obje selekcije pregledavaju cijeli skup podataka relacije.

Pravilo kombiniranja selekcija nemaju utjecaja na vrijeme izvršavanja upita ukoliko se jedna selekcija odvija brzo, iz razloga jer relacija na koju je primijenjena koristi pomoćne strukture podataka (*indekse* i slično), a druga *selekcija* se odvija sporo jer će rezultirajuća selekcija također odvijati sporo.

Bez primjene optimizacije, kod upita $\sigma_{F2}(\sigma_{F1}(r))$, prvo se slogovi reduciraju prema uvjetu F_1 pregledavanjem svih slogova relacije r, da bi se nakon toga taj reducirani skup još jednom reducirao prema uvjetu F_2 što dovodi do redundantnih provjera koje je moguće izvršiti istovremeno te na taj način smanjiti vrijeme izvršavanja upita.

Primjenom optimizacije, kod upita $\sigma_{F}(r)$, uklonjene su redundantne provjere te se jednim prolaskom cijele relacije dobiva traženi rezultat upita.

Dokaz 1. Kombiniranje selekcija ($\sigma_{F2}(\sigma_{F1}(r)) \equiv \sigma_F(r)$)

$\sigma_{F2}(\sigma_{F1}(r))$,

F_1 : (A > 1)

F_2 : (C = 'a')

r	A	B	C
	4	a	f
	6	c	a
	1	d	a
	1	f	c
	3	a	a

$s_1 = \sigma_{F1}(r)$,

F_1 : (A > 1)

$\sigma_{F1}(r)$	A	B	C
	4	a	f
	6	c	a
	3	a	a

$s_2 = \sigma_{F2}(s_1)$,

F_2 : (C = 'a')

$\sigma_{F2}(s_1)$	A	B	C
	6	c	a
	3	a	a

$\sigma_F(r)$,

F : (A > 1) ∧ (C = 'a')

r	A	B	C
	4	a	f
	6	c	a
	1	d	a
	1	f	c
	3	a	a

$s_1 = \sigma_F(r)$,

F : (A > 1) ∧ (C = 'a')

σ_F(r)	A	B	C
	6	c	a
	3	a	a

Vidljivo je da se kombiniranjem selekcija smanjuje broj operacija, a samim time i relacija koje je potrebno izračunati da bi dobili odgovor na postavljeni upit.
Smanjenje broja operacija koje je potrebno izračunati ima direktan utjecaj na optimizaciju upita, odnosno smanjenje vremena potrebnog da se upit izvrši.

8.2.2 Izvlačenje selekcije ispred spoja ili produkta

„Ako uvjet *F* sadrži samo atribute od relacije *r*, a ne one od relacije *s*, tada vrijedi :

$$\sigma_F(r \bowtie_{X=Y} s) \equiv \sigma_F(r) \bowtie_{X=Y} s$$

$$\sigma_F(r \otimes s) \equiv \sigma_F(r) \otimes s$$

(*Baze podataka skripta*, R. Manger, 2003. , 32. str.)

Izvlačenjem selekcije ispred spoja ili produkta smanjuje se vrijeme potrebno za izvršavanje upita. Ukoliko detaljnije pogledamo definirane ekvivalencije tada je jasno vidljivo da se vrijeme izvršavanja upita skraćuje ukoliko se operacije spoja odnosno produkta, kao vremenski „skupe“ operacije, primjene na manji skup podataka.

Bez primjene optimizacije, kod upita $\sigma_F(r \bowtie_{X=Y} s)$, prvo se izvršava operacija spoja između definiranih relacija *r* i *s*, da bi se nakon toga mogla izvršiti operacija selekcije na rezultantnoj relaciji te došlo do traženog rezultata upita. Isto vrijedi i za operaciju Kartezijevog produkta.

Primjenom optimizacije, kod upita $\sigma_F(r) \bowtie_{X=Y} s$, operacija spoja se izvršava između reducirane relacije *r* te relacije *s*. Upravo reduciranjem relacija *r* postiže se smanjenje vremena izvršavanja upita jer se smanjuje broj slogova koji sudjeluju u operaciji spoja, za koju smo rekli da je vremenski „skupa“. Isto vrijedi i za operaciju Kartezijevog produkta.

Pravilo izvlačenja selekcije ispred spoja ili produkta može značajno smanjiti broj n-torki (slogova) koje sudjeluju u operaciji spoja ili produkta.

„Općenitije, ako *F* rastavimo na $F = F_r \wedge F_s \wedge F_c \wedge F'$, gdje F_r sadrži samo atribute od relacije *r*, F_s sadrži samo atribute od relacije *s*, F_c sadrži zajedničke atribute od relacija *r* i *s*, te *F'* predstavlja ostatak od *F*, tada vrijedi :

$$\sigma_F(r \bowtie_{X=Y} s) \equiv \sigma_{F'}(\sigma_{F1}(r) \bowtie_{X=Y} \sigma_{F2}(s)),$$

gdje je $F_1 : F_R \wedge F_C$, a $F_2 = F_S \wedge F_C$.“

(*Baze podataka skripta*, R. Manger, 2003. , 32. str.)

Ekvivalencija za Kartezijev produkt može se definirati na sličan način.

Dokaz 2. Izvlačenje selekcije ispred spoja ili produkta ($\sigma_F(r \bowtie_{r.B = s.B} s) \equiv \sigma_F(r) \bowtie_{r.B = s.B} s$)

$\sigma_F(r \bowtie_{r.B = s.B} s)$,

F : (B = 'a') ∧ (C != 'a')

r	A	B	C
	1	a	b
	7	b	a
	5	c	a

s	B	D	E
	a	1	2
	a	3	9
	c	2	7

$s_1 = r \bowtie_{r.B = s.B} s$

r ⋈ r.B = s.B s	A	B	C	D	E
	1	a	b	1	2
	1	a	b	3	9
	5	c	a	2	7

$s_2 = \sigma_F(s_1)$,

F : (B = 'a') ∧ (C != 'a')

r ⋈ s	A	B	C	D	E
	1	a	b	1	2
	1	a	b	3	9

$\sigma_F(r) \bowtie_{r.B = s.B} s$,

F : (B = 'a') ∧ (C != 'a')

r	A	B	C
	1	a	b
	7	b	a
	5	c	a

s	B	D	E
	a	1	2
	a	3	9
	c	2	7

$s_1 = \sigma_F(r)$,

F : (B = 'a') ∧ (C != 'a')

σ_F(r)	A	B	C
	1	a	b

$s_2 = s_1 \bowtie_{r.B = s.B} S$

$S_1 \bowtie_{r.B = s.B} S$	A	B	C	D	E
	1	a	b	1	2
	1	a	b	3	9

Iz primjera je vidljivo da se izvlačenjem selekcije ispred spoja smanjuje broj slogova relacija koje sudjeluju u operaciji spoja. Spoj je vrlo skupa operacija pa se iz tog razloga pokušava smanjiti broj slogova koji će biti povezivani.
Smanjenjem broja slogova koji sudjeluju u operaciji spoja ima direktan utjecaj na optimizaciju upita, odnosno smanjenje vremena potrebnog da se upit izvrši.

Cijeli postupak vrijedi i za Kartezijev produkt.

8.2.3 Izvlačenje selekcije ispred projekcije

„Ukoliko uvjet *F* sadrži samo projicirane atribute *X*, tada vrijedi :

$$\sigma_F(\Pi_X(r)) \equiv \Pi_X(\sigma_F(r))$$“

(*Baze podataka skripta*, R. Manger, 2003. , 33. str.)

Izvlačenjem selekcije ispred projekcije smanjuje se vrijeme potrebno za izvršavanje upita. Projekcija može povećati trajanje izvršavanja upita zbog eliminacije „duplikata“ slogova. Primjenom ovog pravila optimizacije, to se optimizira na način da se uvjet selekcije provjerava samo na skupu atributa koji su sadržani u uvjetu.
„To je posebno preporučljivo kad postoje fizička sredstva za brzu selekciju.“
(*Baze podataka skripta*, R. Manger, 2003. , 33. str.)

Dokaz 3. Izvlačenje selekcije ispred projekcije ($\sigma_F(\Pi_{A,B,C}(r)) \equiv \Pi_{A,B,C}(\sigma_F(r))$)

$\sigma_F(\Pi_{A,B,C}(r))$,

F : (A > 1) ∧ (B != 'a') ∧ (C = 'a')

r	A	B	C	D
	9	c	a	7
	3	d	a	4
	1	f	c	2
	5	a	a	3

$s_1 = \Pi_{A,B,C}(r)$

$\Pi_{A,B,C}$ (r)	A	B	C
	9	c	a
	3	d	a
	1	f	c
	5	a	a

$s_2 = \sigma_F(s_1)$,

F : A > 1 ∧ B != 'a'

$\sigma_F(s_1)$	A	B	C
	9	c	a
	3	d	a

$\Pi_{A,B,C}(\sigma_F(r))$,

F : (A > 1) ∧ (B != 'a') ∧ (C = 'a')

r	A	B	C	D
	9	c	a	7
	3	d	a	4
	1	f	c	2
	5	a	a	3

$s_1 = \sigma_F(r)$,

F : (A > 1) ∧ (B != 'a') ∧ (C = 'a')

$\sigma_F(r)$	A	B	C	D
	9	c	a	7
	3	d	a	4

$s_2 = \Pi_{A,B,C}(s_1)$

$\Pi_{A,B,C}(s_1)$	A	B	C
	9	c	a
	3	d	a

Vidljivo je da se izvlačenjem selekcije ispred projekcije smanjuje broj atributa na koje se primjenjuje operator selekcije. Na taj način se ostvaruje da relacije, prije same operacije selekcije, sadrži samo one atribute koji se pojavljuju u uvjetu selekcije.

Smanjenjem broja atributa koji sudjeluju u operaciji selekcije ima direktan utjecaj na optimizaciju upita, odnosno smanjenje vremena potrebnog da se upit izvrši.

8.2.4 Kombiniranje projekcija

„Ako su *X*, *Y* i *Z* atributi relacije *r*, tada vrijedi :

$$\Pi_{X}(\Pi_{X,Y}(\Pi_{X,Y,Z}(r))) \equiv \Pi_{X}(r)$$“

(*Baze podataka skripta*, R. Manger, 2003. , 33. str.)

Kombiniranjem projekcija smanjuje se vrijeme potrebno za izvršavanje upita.
U prikazanom primjeru, umjesto 3 projekcije dovoljna je samo jedna. Kako je projekcija vremenski „skupa“ operacija zbog eliminacije „duplikata“ slogova, ovim reduciranjem se postiže značajan doprinos prilikom optimizacije upita.
Pravilo kombiniranja projekcija je zbog jednostavnosti prethodnog primjera vrlo lako uočljivo, ali kod dugačkih, složenih te kompliciranih izraza nije tako jednostavno uočiti redundantne projekcije. Iz tog razloga je vrlo važno da ovo pravilo optimizacije upita bude primijenjeno na korektan način.

Dokaz 4. Kombiniranje projekcija ($\Pi_X(\Pi_{X,Y}(\Pi_{X,Y,Z}(r))) \equiv \Pi_X(r)$)

$\Pi_{A}(\Pi_{A,B}(\Pi_{A,B,C}(r))$

r	A	B	C	D
	9	c	a	7
	3	d	a	4
	1	f	c	2

$s_1 = \Pi_{A,B,C}(r)$

$\Pi_{A,B,C}$(r)	A	B	C
	9	c	a
	3	d	a
	1	f	c

$s_2 = \Pi_{A,B}(s_1)$

$\Pi_{A,B}(s_1)$	A	B
	9	c
	3	d
	1	f

$s_3 = \Pi_A(s_2)$

$\Pi_A(s_2)$	A
	9
	3
	1

$\Pi_A(r)$

r	A	B	C	D
	9	c	a	7
	3	d	a	4
	1	f	c	2

$s_1 = \Pi_A(r)$

Π_A(r)	A	B	C	D
	9	c	a	7
	3	d	a	4
	1	f	c	2

Smanjenjem broja operacija selekcije s tri na jednu ima direktan utjecaj na optimizaciju upita, odnosno smanjenje vremena potrebnog da se upit izvrši.

8.2.5 Izvlačenje projekcije ispred spoja

Vrlo je važno voditi računa da projiciranjem ne izbacimo zajednički/e atribut/e iz relacija prije nego što je bila obavljena operacija spoja.
„Ako *X* označava zajedničke atribute relacija *r* i *s*, tada je :

$$\Pi_X(r \bowtie s) \equiv \Pi_X(r) \bowtie \Pi_X(s)$$“

(*Baze podataka skripta*, R. Manger, 2003. , 33. str.)

Izvlačenjem projekcije ispred spoja smanjuje se vrijeme potrebno za izvršavanje upita na način da se smanjuje broj n-torki (slogova) koje sudjeluju u operaciji spoja. Ovo pravilo ne mora uvijek biti korisno, jer u nekim slučajevima projekcija će spriječiti efikasnu implementaciju spoja. „Primjerice, projekcija može stvoriti privremenu relaciju na koju nisu primjenjive postojeće pomoćne fizičke strukture.“ (*Baze podataka skripta*, R. Manger, 2003. , 33. str.)

Dokaz 5. Izvlačenje projekcije ispred spoja

$(\Pi_{B,C}(r \bowtie_{(r.B = s.B) \wedge (r.C = s.C)} s) \equiv \Pi_{B,C}(r) \bowtie_{(r.B = s.B) \wedge (r.C = s.C)} \Pi_{B,C}(s))$

$\Pi_{B,C}(r \bowtie_{(r.B = s.B) \wedge (r.C = s.C)} s)$

r	A	B	C
	1	a	b
	7	b	a
	5	c	a

s	B	C	D
	a	b	2
	a	b	9
	c	a	7

$s_1 = r \bowtie_{(r.B = s.B) \wedge (r.C = s.C)} s$

r ⋈(r.B = s.B) ∧ (r.C = s.C) s	A	B	C	D
	1	a	b	2
	1	a	b	9
	5	c	a	7

$s_2 = \Pi_{B,C}(s_1)$

$\Pi_{B,C}(s_1)$	B	C
	a	b
	a	b
	c	a

$\Pi_{B,C}(r) \bowtie_{(r.B = s.B) \wedge (r.C = s.C)} \Pi_{B,C}(s)$

r	A	B	C
	1	a	b
	7	b	a
	5	c	a

s	B	C	D
	a	b	2
	a	b	9
	c	a	7

$s_1 = \Pi_{B,C}(r)$

$\Pi_{B,C}(r)$	B	C
	a	b
	b	a
	c	a

$s_2 = \Pi_{B,C}(s)$

$\Pi_{B,C}(s)$	B	C
	a	b
	a	b
	c	a

$S_1 \bowtie_{(r.B = s.B) \wedge (r.C = s.C)} S_2$

$S_1 \bowtie_{(r.B = s.B) \wedge (r.C = s.C)} S_2$	B	C
	a	b
	a	b
	c	a

8.2.6 Optimizacija skupovnih operatora

Ponekad se javlja potreba za optimizacijom operacija u kojima sudjeluju skupovni operatori. „U takvim slučajevima koriste se sljedeća pravila :

$$\sigma_F(r \cup s) \equiv \sigma_F(r) \cup \sigma_F(s)$$

$$\sigma_F(r - s) \equiv \sigma_F(r) - \sigma_F(s)$$

$$\Pi_X(r \cup s) \equiv \Pi_X(r) \cup \Pi_X(s)$$

$$\Pi_X(\sigma_{F1}(r)) \cup \Pi_X(\sigma_{F2}(r)) \equiv \Pi_X(\sigma_F(r)),$$

gdje je $F : F_1 \vee F_2$“

(*Baze podataka skripta*, R. Manger, 2003. , 33. str.)

Ekvivalencija $\Pi_X(r \cup s) \equiv \Pi_X(r) \cup \Pi_X(s)$ vrijedi pod pretpostavkom da atribut *X* sadrži ključne atribute relacije *r*, a samim time i ključne atribute od relacije *s*.

Optimizacijom skupovnih operatora smanjuje se vrijeme potrebno za izvršavanje upita na način da se smanjuje broj n-torki (slogova) koje sudjeluju u operacijama unije i razlike.

„Operator presjeka je specijalni slučaj od join, pa za njega vrijede ista pravila kao i za join.“

(*Baze podataka skripta*, R. Manger, 2003. , 33. str.)

Dokaz 6. Optimizacija skupovnih operatora ($\sigma_F(r \cup s) \equiv \sigma_F(r) \cup \sigma_F(s)$)

$\sigma_F(r \cup s)$,

$F : (A > 1) \wedge (C != 9)$

r	A	B	C
	3	4	6
	5	5	9

s	A	B	C
	1	2	9
	7	2	5

$s_1 = r \cup s$

r ∪ s	A	B	C
	3	4	6
	5	5	9
	1	2	9
	7	2	5

$\sigma_F(s_1)$,

F : (A > 1) ∧ (C != 9)

$\sigma_F(s_1)$	A	B	C
	3	4	6
	7	2	5

$\sigma_F(r) \cup \sigma_F(s)$,

F : (A > 1) ∧ (C != 9)

r	A	B	C
	3	4	6
	5	5	9

s	A	B	C
	1	2	9
	7	2	5

$s_1 = \sigma_F(r)$,

F : (A > 1) ∧ (C != 9)

$\sigma_F(r)$	A	B	C
	3	4	6

$s_2 = \sigma_F(s)$,

F : (A > 1) ∧ (C != 9)

$\sigma_F(s)$	A	B	C
	7	2	5

$s_1 \cup s_2$,

F : (A > 1) ∧ (C != 9)

$s_1 \cup s_2$	A	B	C
	3	4	6
	7	2	5

Dokaz 7. Optimizacija skupovnih operatora ($\sigma_F(r - s) \equiv \sigma_F(r) - \sigma_F(s)$)

$\sigma_F(r - s)$

F : B > 1

r	A	B	C
	3	1	7
	2	2	8
	1	9	4

s	A	B	C
	2	2	8
	6	4	1
	3	7	5

$s_1 = r - s$

r – s	A	B	C
	3	1	7
	1	9	4

$\sigma_F(s_1)$,

F : B > 1

$\sigma_F(s_1)$	A	B	C
	1	9	4

$\sigma_F(r) - \sigma_F(s)$

r	A	B	C
	3	1	7
	2	2	8
	1	9	4

s	A	B	C
	2	2	8
	6	4	1
	3	7	5

$s_1 = \sigma_F(r)$,

F : B > 1

σ_F(r)	A	B	C
	2	2	8
	1	9	4

$s_2 = \sigma_F(s)$,

F : B > 1

σ_F(s)	A	B	C
	2	2	8
	6	4	1
	3	7	5

$s_1 - s_2$

$\sigma_F(s_1)$	A	B	C
	1	9	4

Dokaz 8. Optimizacija skupovnih operatora ($\Pi_{A,C,D}(r \cup s) \equiv \Pi_{A,C,D}(r) \cup \Pi_{A,C,D}(s)$)

$\Pi_{A,C,D}(r \cup s)$

r	A	B	C	D
	3	4	6	a
	5	5	9	b

s	A	B	C	D
	1	8	3	c
	2	4	0	f

$s_1 = r \cup s$

r ∪ s	A	B	C	D
	3	4	6	a
	5	5	9	b
	1	8	3	c
	2	4	0	f

$s_2 = \Pi_{A,C,D}(s_1)$

$\Pi_{A,C,D}(s_1)$	A	C	D
	3	6	a
	5	9	b
	1	3	c
	2	0	f

$\Pi_{A,C,D}(r) \cup \Pi_{A,C,D}(s)$

r	A	B	C	D
	3	4	6	a
	5	5	9	b

s	A	B	C	D
	1	8	3	c
	2	4	0	f

$s_1 = \Pi_{A,C,D}(r)$

$\Pi_{A,C,D}$(r)	A	C	D
	3	6	a
	5	9	b

$s_2 = \Pi_{A,C,D}(s)$

$\Pi_{A,C,D}$(s)	A	C	D
	1	3	c
	2	0	f

$s_1 \cup s_2$

$s_1 \cup s_2$	A	C	D
	3	6	a
	5	9	b
	1	3	c
	2	0	f

Dokaz 9. Optimizacija skupovnih operatora ($\Pi_{A,C}(\sigma_{F1}(r)) \cup \Pi_{A,C}(\sigma_{F2}(r)) \equiv \Pi_{A,C}(\sigma_{F}(r))$)

$\Pi_{A,C}(\sigma_{F1}(r)) \cup \Pi_{A,C}(\sigma_{F2}(r))$,

F_1 : (B != 'a')

F_2 : (A = 3)

r	A	B	C
	3	a	6
	5	c	9
	7	e	1
	2	a	3

$s_1 = \sigma_{F1}(r)$,

F_1 : (B != 'a')

$\sigma_{F1}(r)$	A	B	C
	5	c	9
	7	e	1

$s_2 = \Pi_{A,C}(s_1)$

$\Pi_{A,C}(s_1)$	A	C
	5	9
	7	1

$s_3 = \sigma_{F2}(r)$,

F_2 : (A = 3)

$\sigma_{F2}(r)$	A	B	C
	3	a	6

$s_4 = \Pi_{A,C}(s_3)$

$\Pi_{A,C}(s_3)$	A	C
	3	6

$s_2 \cup s_4$

$s_2 \cup s_4$	A	C
	5	9
	7	1
	3	6

$\Pi_{A,C}(\sigma_F(r))$,

F : (B != 'a') ∨ (A = 3)

r	A	B	C
	3	a	6
	5	c	9
	7	e	1
	2	a	3

$s_1 = \sigma_F(r)$,

F : (B != 'a') ∨ (A = 3)

σ_F(r)	A	B	C
	3	a	6
	5	c	9
	7	e	1

$\Pi_{A,C}(s_1)$

$\Pi_{A,C}(s_1)$	A	C
	3	6
	5	9
	7	1

Sva navedena pravila optimizacije odnose se na algebarske izraze, odnosno upite relacijske algebre. Osim pravila koja se direktno primjenjuju na algebarski izraz temeljem definiranih ekvivalencija postoji i nekoliko pravila vezanih uz veličinu operanada, odnosno relacija.

8.2.7 Pravila vezana uz veličinu relacija

U nekim slučajevima, da bi se pravila optimizacije algebarskih izraza mogla primijeniti, potrebno je znati veličinu relacije, odnosno broj slogova relacije koja se koristi u operaciji.
Nepisano pravilo je da se kod spojeva u skupovnih operatora relacija s manjim brojem slogova stavlja na desnu stranu spoja ili skupovnog operatora.
U slučaju da se u nekom algebarskom izrazu niže više istih operatora za redom tada se operator primjenjuje na dvije manje relacije, a tek na kraju s onom koja ima najviše slogova.
Pravila vezana uz veličinu relacija primjenjiva su na gotove sve operatore zbog svojstava komutativnosti i asocijativnosti operatora.
Pravila vezana uz veličinu relacija mogu se podijeliti u dvije skupine :

1. pravila komutativnosti
2. pravila asocijativnosti

Pravila komutativnosti

- $r \bowtie s \equiv s \bowtie r$
- $r \cup s \equiv s \cup r$
- $r \cap s \equiv s \cap r$
- $r \otimes s \equiv s \otimes r$

Pravila asocijativnosti

- $(r \cup s) \cup t \equiv r \cup (s \cup t)$
- $(r \cap s) \cap t \equiv r \cap (s \cap t)$
- $(r \otimes s) \otimes t \equiv r \otimes (s \otimes t)$

8.2.8 Pravila vezana uz selekciju

Ranije je u definiciji selekcije bilo navedeno da se selekcijom smanjuje broj slogova relacije. Jasno je da smanjenje broja slogova neke relacije ima direktan utjecaj na performanse upite. Razlog tome je što smanjenjem broja slogova reduciramo količinu „posla“ koji pojedini opeartor treba obaviti. Zbog toga se selekciju pokušava spustiti što je niže moguće u stablu algebarskih operatora, a da pritom ne narušimo inicijalno djelovanje selekcije.
Pravilo spuštanja selekcije što je niže moguće poznato je pod nazivom uvlačenje selekcije.
„Prve verzije *query optimizera* smatrale su pravilo uvlačenja selekcije primarnom strategijom poboljšanja logičkog plana upita.“
(prema *Database System The Complete Book*, Garcia Molina H. i suradnici, 2009. , 770. str.)

Ukoliko se uvlačenje selekcije primjenjuje na neki od binarnih operatora (Kartezijev produkt, unija, presjek, razlika, spojevi) tada vrijede sljedeća pravila :

1. kod unije je bitno da se selekcija „uvuče“ u oba operanda koja sudjeluju u operaciji, odnosno vrijedi $\sigma_F(r \cup s) \equiv \sigma_F(r) \cup \sigma_F(s)$
2. kod razlike je bitno da se selekcija „uvuče“ u prvi operand ili opcionalno u drugi operand, odnosno vrijedi $\sigma_F(r - s) \equiv \sigma_F(r) - s$
3. ostale operacije zahtijevaju da se selekcija „uvuče“ samo u jedan od operanada na koji se uvjet selekcije može primijeniti, odnosno vrijedi $\sigma_F(r \otimes s) \equiv \sigma_F(r) \otimes s$

Kada govorimo o uvlačenju selekcije tada je potrebno istaknuti neke specijalne slučajeve kao što su djelovanje operatora ne praznu relaciju, selekcije i spojevi kod kojih su uvjeti uvijek *true* ili uvijek *false* te selekcija projekcije koja izvlači sve atribute relacije.
U takvim slučajevima vrijede sljedeća pravila :

- svaka selekcija koja djeluje na praznu relaciju kao rezultat vraća praznu relaciju
- ako je uvjet selekcije u nekom algebarskom izrazu uvijek *true*, tada selekcija kao rezultat vraća istu relaciju, odnosno vrijedi $\sigma_F(r) = r$
- ako je jedna od relacija na koje djeluje skupovni operator unije prazna, tada unija kao rezultat vraća drugu relaciju, odnosno ako je relacije r prazna tada vrijedi $r \cup s = s$

8.3 Primjer optimizacije upita

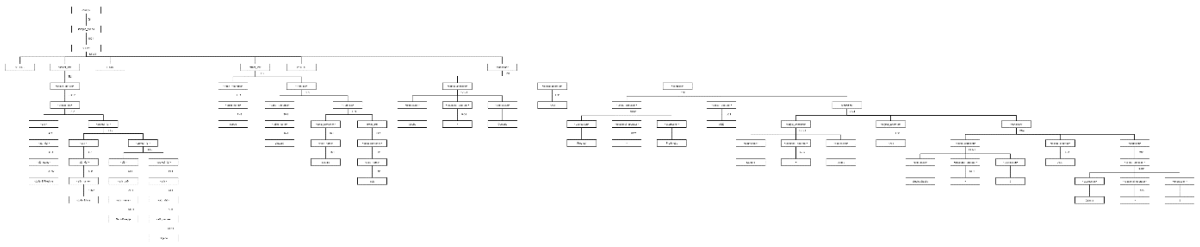
Primjer 34. Optimizacija upita (*Obrada upita* (prezentacija), 2019. , A. Lovrenčić)
Pretpostavimo da koristimo bazu podataka fakulteta u kojoj su definirane relacije *osobe*, *studenti*, *predmeti* te *ispiti*.

Neka je zadan upit *U :*
Pronađi prezime, imena, naziv kolegija te ocjenu studenata 3. godine koji su položili kolegije, odnosno koji su ostvarili ocjenu veću od 1.

SQL(U) :

```
SELECT      studenti.Prezime, stundeti.Ime, NazivKolegija, Ocjena
FROM        osobe, studenti, kolegiji, ispiti
WHERE       Osoba = OsobaId
AND         Predmet = PredmetId
AND         Student = Osoba
AND         GodinaStudija = 3
AND         Ocjena > 1
```

Primjetimo da se u primjeru koristi stariji način povezivanja relacija kod kojeg se relacije povezuju *WHERE* klauzulom na temelju zajedničkih atributa.
U novije vrijeme je taj način zamijenjen spajanjem korištenjem *INNER JOIN* klauzule.



Slika 6. Stablo parsiranja Primjer 34.

Na temelju stabla parsiranja *query preprocessor* bi preveo upit u relacijsku algebru.

RA(U): $\Pi_{studenti.Prezime,studenti.Ime,NazivKolegija,Ocjena}(\sigma_F(osobe \otimes studenti \otimes kolegiji \otimes ispiti))$

$F : (Osoba = OsobaId) \wedge (Predmet = PredmetId) \wedge (Student = Osoba)$
$\wedge (GodinaStudija = 3) \wedge (Ocjena > 1)$

Kreirani upit relacijske algebre je vrlo složen te ga je potrebno optimizirati kako bi novokreirani upitni plan bio efikasniji, odnosno kako bi vrijeme izvršavanja upita bilo kraće.

1. Uvlačenje selekcije

U kreiranom algebraskom izrazu uvjet *Ocjena > 1* odnosi se samo na relaciju *ispiti*, pa se upit može optimizirati na način da se uvjet redefinira u selekciju relacije *ispiti*. Na taj način se selekcija izvršava nad manjim skupom podataka te je samim time efikasnija u kontekstu vremena izvršavanja operacija.

RA(U): $\Pi_{studenti.Prezime,studenti.Ime,NazivKolegija,Ocjena}(\sigma_F(osobe \otimes studenti \otimes kolegiji \otimes \sigma_{Ocjena>1}(ispiti)))$

F : (Osoba = OsobaId) ∧ (Predmet = PredmetId) ∧ (Student = Osoba) ∧ (GodinaStudija = 3)

2. Uvlačenje selekcije

U novom algebraskom izrazu uvjet *GodinaStudija = 3* odnosi se samo na relaciju *studenti*, pa se upit može optimizirati na način da se taj uvjet redefinira u selekciju relacije *studenti*. Na taj način se selekcija izvršava nad manjim skupom podataka te je samim time efikasnija u kontekstu vremena izvršavanja operacija.

RA(U): $\Pi_{studenti.Prezime,studenti.Ime,NazivKolegija,Ocjena}$

$(\sigma_F(osobe \otimes \sigma_{GodinaStudija=3}(studenti) \otimes kolegiji \otimes \sigma_{Ocjena>1}(ispiti)))$

F : (Osoba = OsobaId) ∧ (Predmet = PredmetId) ∧ (Student = Osoba)

3. Uvlačenje selekcije

U novom algebraskom izrazu uvjet *Student = Osoba* odnosi se samo na relacije *studenti* i *ispiti*, pa se upit može optimizirati na način da se taj uvjet redefinira u selekciju. Na taj način se selekcija izvršava nad manjim skupom podataka te je samim time efikasnija u kontekstu vremena izvršavanja operacija.

RA(U): $\Pi_{studenti.Prezime,studenti.Ime,NazivKolegija,Ocjena}$

$(\sigma_F(osobe \otimes \sigma_{Student=Osoba}(\sigma_{GodinaStudija=3}(studenti) \otimes \sigma_{Ocjena>1}(ispiti)) \otimes kolegiji))$

F : (Osoba = OsobaId) ∧ (Predmet = PredmetId)

4. Uvlačenje selekcije

U novom algebraskom izrazu uvjet *Predmet* = *PredmetId* odnosi se samo na relacije *predmeti* i *ispiti*, pa se upit može optimizirati na način da se taj uvjet redefinira u selekciju. Na taj način se selekcija izvršava nad manjim skupom podataka te je samim time efikasnija u kontekstu vremena izvršavanja operacija.

RA(U): $\Pi_{studenti.Prezime,studenti.Ime,NazivKolegija,Ocjena}$

$(\sigma_{F}(osobe \otimes \sigma_{Predmet=PredmetId}\ (\sigma_{Student=Osoba}(\sigma_{GodinaStudija=3}(studenti) \otimes \sigma_{Ocjena>1}(ispiti)) \otimes kolegiji)))$

F : (Osoba = OsobaId)

5. Kombiniranje selekcija

U novom algebraskom izrazu uvjete *Student* = *Osoba*, *GodinaStudija* = *3* i *Ocjena* > *1* možemo spojiti prirodnim spojem te na taj način optimizirati upit.

RA(U): $\Pi_{studenti.Prezime,studenti.Ime,NazivKolegija,Ocjena}$

$(\sigma_{F}(osobe \otimes \sigma_{Predmet=PredmetId}((studenti \bowtie_{(Student=Osoba) \wedge (GodinaStudija=3) \wedge (Ocjena>1)} ispiti) \otimes kolegiji)))$

F : (Osoba = OsobaId)

6. Pretvorba Kartezijevog produkta u spoj

U novom algebraskom izrazu Kartezijeve produkte možemo pretvoriti u unutarnje spojeve te na taj način optimizirati upit.

RA(U): $\Pi_{studenti.Prezime,studenti.Ime,NazivKolegija,Ocjena}$

$(osobe \bowtie_{(Osoba=OsobaId)} (studenti \bowtie_{(Student=Osoba) \wedge (GodinaStudija=3) \wedge (Ocjena>1)} ispiti)$

$\bowtie_{(Predmet=PredmetId)} kolegiji)$

Novi algebarski izraz je optimalan jer zadovoljava uvjet minimalnosti, odnosno na dobiveni izraz ne može se primijeniti niti jedno od pravila optimizacije upita. Optimizirani upit predstavlja fizički plan upita koji će se zatim izvršavati unutar *SUBP-a.*

Primjer 35. Optimizacija upita

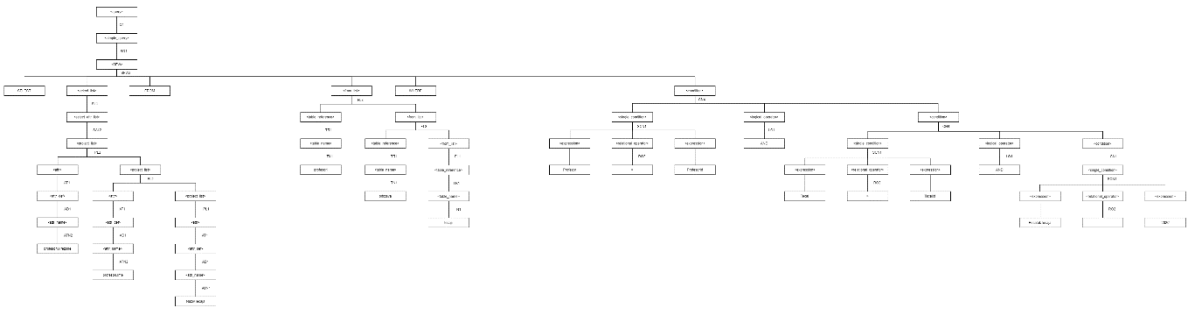
Pretpostavimo da koristimo bazu podataka glazbenog fakulteta u kojoj su definirane relacije *profesori*, *odrzava*, *tecajevi*.

Neka je zadan upit *U :*

Pronađi prezime, imena profesora te nazive tečajeva koje ti profesori održavaju u 2020. godini.

SQL(U) :

```
SELECT      profesori.Prezime, profesori.Ime, NazivTecaja
FROM        profesori, odrzava, tecajevi
WHERE       Profesor = ProfesorId
AND         Tecaj = TecajId
AND         PocetakTecaja = '2020'
```



Slika 7. Stablo parsiranja Primjer 35.

Na temelju stabla parsiranja *query preprocessor* bi preveo upit u relacijsku algebru.

RA(U): $\Pi_{\text{profesori.Prezime, profesori.Ime,NazivKolegija}}(\sigma_F(\textit{profesori} \otimes \textit{odrzava} \otimes \textit{tecajevi}))$

F : (Profesor = ProfesorId) ∧ (Tecaj = TecajId) ∧ (PocetakTecaja = '2020')

Kreirani upit relacijske algebre je vrlo složen te ga je potrebno optimizirati kako bi novokreirani upitni plan bio efikasniji, odnosno kako bi vrijeme izvršavanja upita bilo kraće. Optimizirani upit predstavlja fizički plan upita koji će se zatim izvršavati unutar *SUBP-a.*

1. Uvlačenje selekcije

U kreiranom algebraskom izrazu uvjet PocetakTecaja = *'2020'* odnosi se samo na relaciju *odrzava*, pa se upit može optimizirati na način da se uvjet redefinira u selekciju relacije *odrzava*. Na taj način se selekcija izvršava nad manjim skupom podataka te je samim time efikasnija u kontekstu vremena izvršavanja operacija.

RA(U): $\Pi_{\text{profesori.Prezime, profesori.Ime,NazivKolegija}}(\sigma_{F}(\textit{profesori} \otimes \textit{odrzava}_{\text{PocetakTecaja='2020'}} \otimes \textit{tecajevi}))$

$$F : (\text{Profesor} = \text{ProfesorId}) \wedge (\text{Tecaj} = \text{TecajId})$$

2. Pretvorba Kartezijevog produkta u spoj

U novom algebraskom izrazu Kartezijeve produkte možemo pretvoriti u unutarnje spojeve te na taj način optimizirati upit.

RA(U): $\Pi_{\text{profesori.Prezime, profesori.Ime,NazivKolegija}}$

$(\textit{profesori} \bowtie_{\text{(Profesor=ProfesorId)}} \textit{odrzava} \bowtie_{\text{(Tecaj=TecajId)} \wedge \text{(PocetakTecaja='2020')}} \textit{tecajevi}))$

Novi algebarski izraz je optimalan jer zadovoljava uvjet minimalnosti, odnosno na dobiveni izraz ne može se primijeniti niti jedno od pravila optimizacije upita.

9. Zaključak

Relacijska algebra podrazumijeva operacije definirane nad relacijama te podacima koji se nalaze unutar samih relacija. Relacijska algebra se svodi na računanje vrijednosti različitih algebarskih izraza koji mogu biti građeni od unarnih te binarnih operacija, operanada te zagrada. Kada govorimo o binarnim operacijama tada podrazumijevamo da ta operacija uzima podatke iz nekog skupa te im nakon operacije pridružuje neki podataka koji je iz tog istog skupa. Relacijska algebra ima veliku primjenu prilikom postavljanja upita nad bazom podataka. Osnove relacijske algebre uveo je Edgar Codd u svojim radovima iz 70-ih godina 20. stoljeća.

Relacijski operatori od kojih je građena klasična relacijska algebra su: skupovni operatori (unija (∪), presjek (∩), razlika (**–**)) , projekcija (⊓) , selekcija (σ) , prirodno spajanje (⋈) , preimenovanje atributa (δ) , Kartezijev produkt (⊗) , aktivni komplement (AC) te kvocijent (÷). Navedeni operatori relacijske algebre su primijenjeni unutar sustava za upravljanje bazom podataka (*SUBP*) na sljedeći način. Prilikom postavljanja upita, unutar naredbe *SELECT*, zadana su određena ograničenja nad podacima koji će biti vraćeni kao odgovor na upit. Zadana ograničenja predstavljaju operatore relacijske algebre. U tom pogledu, sama *SELECT* klauzula predstavlja operator projekcije (⊓), klauzula *FROM* predstavlja operator produkta, odnosno kartezijevog produkta (⊗), klauzule *WHERE* i *JOIN* predstavljaju operator selekcije (σ), klauzule *UNION*, *INTERSECT* i *EXCEPT*/*MINUS* predstavljaju operatore (∪, ∩, **–**) respektivno.

Relacijski operatori ne moraju se koristiti striktno pojedinačno, već se mogu međusobno kombinirati što ima veliku primjenu prilikom postavljanja složenih upita nad bazom podataka.
Relacijski operatori imaju definiran prioritet.
Prioritet operatora od najvećeg prema najmanjem je redom:

1. projekcija
2. selekcija
3. produkt
4. prirodno spajanje / kvocijent
5. razlika
6. presjek / unija

Logičke operacije, *NOT, AND* i *OR*, također imaju definiran prioritet. Prioritet logičkih operatora od najvećeg prema najmanjem je redom :

1. NOT
2. AND
3. OR

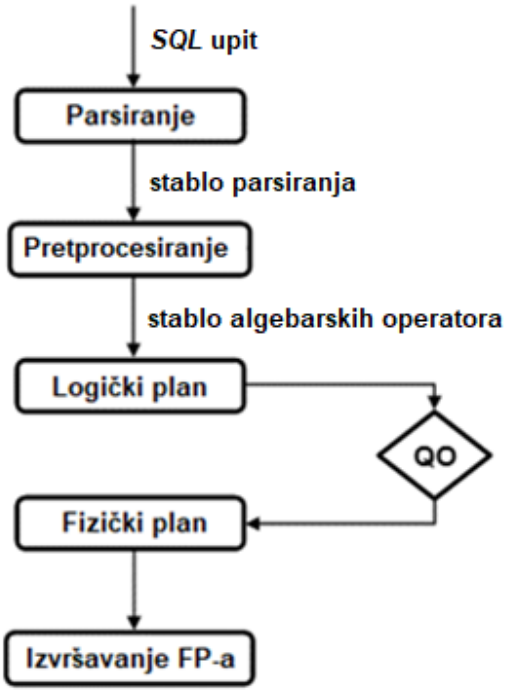
Prioritet operatora je važan prilikom složenih upita kod kojih je potrebno odrediti redoslijed djelovanja pojedinog operatora.

Svaki upit nad bazom podataka se iz deklarativnog jezika pretvara u upit relacijske algebre te nakon toga izvršava. Naime, sintaksa relacijske algebre je jednostavnija te čišća od *SQL-a* i drugih deklarativnih upitnih jezika zbog čega je algoritmi jednostavnije interpretiraju. Međutim, klasična relacijska algebra je skupovno orijentirana, dok je *SQL* multiskupovno orijentiran zbog čega dolazi do neusklađenosti definicija. Kako bi se taj problem razriješio uvedena je proširena relacijska algebra.

Proširena relacijska algebra redefinira klasičnu relacijsku algebru kako bi ona bila usklađena s deklarativnim upitnim jezicima. Tu se prije svega misli na uvođenje dodatnih operatora te prelazak na multiskupovnu orijentaciju. Na taj način je omogućeno da se svaki *SQL* upit može prevesti u korespondentan upit u sintaksi relacijske algebre.

Prevođenje *SQL* upita u upit relacijske algebre u suštini obavlja *parser* komponenta koja kreira stablo parsiranja na temelju kojeg *query preprocessor* kreira stablo algebarskih operatora, odnosno upit u sintaksi relacijske algebre. Prilikom prevođenja upita vrlo važnu ulogu ima i *query optimizer* komponenta koja na temelju upita koji dobiva od *parsera* generira alternativne planove te odabire onaj koji je najefikasniji. Upitni plan se sastoji od sekvence operacija koje će biti izvršene redoslijedom kojim su navedene. Redoslijed operacija je vrlo važan, jer samo promjenom redoslijeda operacija može se postići značajna optimizacija upita.

Postupak obrade upita može se jednostavno prikazati sljedećom slikom.



Slika 8. Proces obrade upita

10. Popis literature

[1] Carić T. , Buntić M. (2015) *Uvod u relacijske baze podataka*. Zagreb. Preuzeto 24. lipnja 2020. s http://files.fpz.hr/Djelatnici/tcaric/Tonci-Caric-Baze-podataka.pdf

[2] Center for Bioinformatics and Computational Biology *Relational algebra*. Preuzeto 29. lipnja 2020. s http://www.cbcb.umd.edu/confcour/Spring2014/CMSC424/Relational_algebra.pdf

[3] Codd E. (1972) *A Relational Model of Data form Large Shared Data Banks*. Preuzeto 28. Lipnja 2020. s https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf

[4] Garcia-Molina H. i suradnici (2009) *Database Systems The Complete Book.* London: Pearson Education Inc.

[5] Lovrenčić A. *Obrada upita* (prezentacija s predavanja na FOBP). Preuzeto 1. srpnja 2020. s https://elf.foi.hr/pluginfile.php/109650/mod_resource/content/1/PDD11-HR1.pdf

[6] Maleković M. , Schatten M. (2017) *Teorija i primjena baza podataka* (udžbenik Sveučilišta u Zagrebu). Varaždin: TIVA-FOI

[7] Maleković M. , Rabuzin K. (2016) *Uvod u baze podataka*. Varaždin: Mini-Print-Logo d.o.o.

[8] Manger R. (2014) *Baze podataka* (udžbenik Sveučilišta u Zagrebu). Zagreb: ELEMENT

[9] Manger R. (2003) *Baze podataka* (skripta). Zagreb. Preuzeto 23. lipnja 2020. s http://jadran.izor.hr/~dadic/EKO/baze-podataka.pdf

[10] Molkova L. (2009) *Relational Algebra Expression Evaluation*. Češka. Preuzeto 30. lipnja 2020. s https://is.muni.cz/th/eifag/bc_thesis.pdf

[11] Ramakrishnan R. , Gehrke J. (2003) *Database Management Systems: Third Edition*. New York: McGraw-Hill

[12] Schatten M. *Relacijski operatori*(prezentacija s predavanja na BP1). Preuzeto 23. lipnja 2020. s https://elfarchive1819.foi.hr/mod/resource/view.php?id=26952

[13] Schatten M. *Model podataka i pojam relacije* (prezentacija s predavanja na BP1). Preuzeto 26. lipnja 2020. s https://elfarchive1819.foi.hr/mod/resource/view.php?id=26930

11. Popis slika