Izražajna snaga jezika SQL

Bogdanić, Antonela

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:697078

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Antonela Bogdanić

IZRAŽAJNA SNAGA JEZIKA SQL

Diplomski rad

Voditelj rada:
doc. dr. sc. Marko Horvat

Zagreb, studeni 2022.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Zahvaljujem obitelji na potpori i "vjetru u leđa" svih ovih godina.
Hvala prijateljima za stvaranje uspomena, kako na fakultetu, tako i izvan njega.
Zahvaljujem mentoru doc. dr. sc. Marku Horvatu na pomoći, trudu i vremenu uloženom u ovaj rad.

Sadržaj

Sadržaj **iv**

Uvod **2**

1 Relacijska algebra **3**
1.1 Relacijski model 3
1.2 Relacijska algebra 5

2 SQL **11**
2.1 SQL kao jezik za postavljanje upita 11
2.2 Verzije SQL-a 21
2.3 SQL3 i rekurzija 22

3 Izražajna snaga SQL-a **29**
3.1 Relacijska algebra i SQL 29
3.2 Proširenje relacijske algebre 30
3.3 Lokalnost SQL upita 34
3.4 Logika i SQL 39
3.5 Zaključak 43
3.6 Ograničenja rekurzije 44

Bibliografija **49**

Uvod

Izloženi smo značajnoj količini informacija koje se nalaze oko nas. Svakog dana sudjelujemo u stvaranju, dijeljenju i spremanju informacija, ali i korištenju onih koje su već negdje pohranjene. Baze podataka su mjesta na koja spremamo podatke nad kojima se kasnije vrše izračuni ili upiti. S obzirom na veličinu i složenost baza podataka, korištenje informacija koje su pohranjene unutar njih bilo bi neučinkovito i zahtjevno bez specijaliziranog softvera. Sustav za upravljanje bazama podataka (kraće DBMS prema eng. *Database Management System*) je posrednik između baze podataka i klijenta (najčešće aplikacije). DBMS mora [12] klijentu omogućiti definiranje nove baze i njezine sheme, pohranu velike količine podataka, načine održavanja i sigurnost, stvaranje i modificiranje podataka te postavljanje upita nad podacima. Navedeni zadaci obavljaju se koristeći jezike za rad s bazama podataka. Tradicionalna podjela [21] tih jezika obuhvaća jezike za opis podataka, jezike za manipuliranje podacima i jezike za postavljanje upita. Svi navedeni jezici su programski jezici, no ne spadaju u programske jezike opće namjene. Naime, od njih se ne očekuje da budu Turing-potpuni niti služe za izradu aplikacija.

U ovom diplomskom radu ćemo se baviti relacijskim bazama podataka i jezikom SQL koji objedinjuje svojstva sve tri navedene vrste jezika za rad s bazama. Iako je SQL primjer integriranog jezika, nas će u ovome radu više zanimati SQL kao jezik za postavljanje upita. Upite postavljamo nad podacima u bazi, a odgovori na upit su podaci iz baze, zaključci ili izračuni koji koriste podatke iz baze. Jezici za postavljanje upita moraju zadovoljavati sljedeće uvjete [5]: dobra tipiziranost (eng. *well-typedness*), učinkovito izračunavanje (eng. *effective computability*) te generičnost (eng. *genericity*). Takvi uvjeti često utječu na izražajnost jezika. Granice izražajnosti zapravo određuju skup dozvoljenih upita, ali isto tako određuju koje ideje i koncepti jesu, odnosno nisu izrazivi u jeziku. Na primjer, možemo proučavati dozvoljenost rekurzivnih upita u SQL-u kojima možemo izraziti razne koncepte i ideje u jeziku. Primjerice, koncept rješenja igre sudoku moguće je izraziti upitom [2] u SQL3 standardu koristeći rekurziju. Međutim, problem dohvatljivosti (eng. reachability), vezan uz provjeru postojanja puta među vrhovima, nije izraziv u tada posljednjem standardu jezika SQL, poznatijem pod nazivom SQL2. Navedena nemogućnost iskazivanja rekurzivnih upita bit će razlog uvođenja rekurzije u SQL.

Nastavno na primjere, nas će zanimati izražajnost SQL-a pa ćemo proučiti koje upite

možemo, a koje ne možemo postaviti tim jezikom za postavljanje upita. Stoga ćemo pregledom jezika dobiti uvid u način i ograničenja zadavanja upita u tom jeziku. Obzirom na to da se granice izražajnosti relacijske algebre često (krivo) koriste za postavljanje granica izražajnosti SQL-a, definirat ćemo i operatore relacijske algebre i proučiti odnos između izražajnosti tih jezika.

Za početak, u prvom poglavlju ćemo definirati relacijsku algebru i opisati operatore na jednostavnom primjeru. Za to ćemo se trebati podsjetiti pojmova poput relacija, n-torka i atribut. U drugom ćemo poglavlju opisati jezik SQL te njegove standarde koristeći primjer iz prethodnog poglavlja. Detaljnije ćemo opisati standard SQL-a iz 1999. godine poznatiji kao SQL3, s posebnim naglaskom na rekurzivne upite tada uvedene u jezik. U trećem i posljednjem poglavlju cilj nam je ispitati izražajnost SQL-a. Pritom ćemo koristiti metodu uspoređivanja izražajnosti jezika te važno svojstvo lokalnosti upita. Naposljetku ćemo se osvrnuti na utjecaj uvođenja rekurzije u jezik SQL na granice njegove izražajnosti.

Uz fizičku kopiju ovog rada priložena je datoteka *BP_primjeri.txt*. U njoj se nalaze SQL naredbe kojima se kreiraju i pune tablice koje koristimo u radu. Također, u datoteci su zapisani i svi SQL upiti iz ovog rada. Može se pronaći i na [3].

Poglavlje 1

Relacijska algebra

Ako želimo proučavati ekspresivnost jezika i odrediti granicu izražajnosti tog jezika, potrebno ga je definirati i/ili opisati. Zato ćemo se u ovom poglavlju detaljnije baviti relacijskom algebrom i njezinim operatorima. Za početak ćemo se prisjetiti pojmova koji su vezani uz relacijski model baza podataka i opisati bazu podataka koju ćemo koristiti kako bismo lakše prikazali djelovanje operatora relacijske algebre.

1.1 Relacijski model

Kako bismo uopće mogli definirati relacijsku algebru i promatrati djelovanje operatora, potrebno je definirati pojmove vezane uz model baze podataka. Model baze podataka je skup pravila koja određuju kako može izgledati logička struktura baze podataka. Model koji ćemo mi koristiti u ovome radu je relacijski model. **Relacijski model** je model baze podataka koji je zasnovan na relacijama, tj. podaci u bazi i veze među njima prezentirani su kao tablice.

Relacija je dvodimenzionalna tablica[1] u koju se pohranjuju podaci. Svaki redak tablice nazivamo ***n*-torka**. Jedinstvenost *n*-torki važno je svojstvo relacije stoga relaciju možemo promatrati i kao skup *n*-torki. Pojam **atributa** poistovjećujemo sa stupcem tablice. U svakom stupcu tablice nalaze se vrijednosti atributa. Sve vrijednosti unutar jednog stupca moraju biti istog tipa, odnosno za pojedini atribut definiramo **domenu atributa** kao skup svih mogućih vrijednosti tog atributa.

Relacijska shema baze podataka je skup shema pojedinih relacija. **Shema relacije** je naziv za ime relacije i skup atributa relacije. U shemi relacije ponekad se dodatno ističu i atributi koje nazivamo ključ. **Ključ relacije** je podskup skupa svih atributa koji osigurava jedinstvenost retka tablice. Naime, različiti redci ne mogu imati iste vrijednosti atributa

[1] U radu ćemo koristiti oba pojma ravnopravno. Češće ćemo kod relacijske algebre govoriti o relacijama, a kod SQL-a o tablicama.

u ključu. Također, definiramo **stupanj relacije** kao broj atributa relacije i **kardinalnost relacije** kao broj n-torki u relaciji. Prethodno opisane pojmove možemo prikazati na konkretnom primjeru 1.1.1.

Primjer 1.1.1. *Opisat ćemo bazu podataka koju ćemo koristiti u radu. Baza podataka vezana je uz knjige, njihove autore i izdavače. Relacijska shema naše baze glasi:*

```
Knjige(ISBN, Ime, ID_autor, ID_izdavac, Godina, Zanr),
Autori(ID_autor, Ime, Prezime),
Izdavaci(ID_izdavac, Ime).
```

Relacije koje imamo u bazi su `Knjige`, `Autori` *te* `Izdavaci`*. Atributi relacija zapisani su unutar zagrada, a stupnjevi relacija su redom* $6, 3, 2$*. Postojeće n-torke unutar svake relacije prikazane su u tablicama 1.1, 1.2, 1.3, pa je kardinalnost relacija redom* $10, 5, 6$*.*

ISBN	Ime	ID_autor	ID_izdavac	Godina	Zanr
9789531975766	Baze podataka	0001	001	2014	obrazovanje
9789531976008	Softversko inzenjerstvo	0001	001	2016	obrazovanje
0009536166119	Prstenova druzina	0002	002	2002	fantastika
0009536166118	Dvije kule	0002	002	2002	fantastika
0009536166135	Povratak kralja	0002	002	2002	fantastika
9789532521849	Crveni krug	0003	003	2015	kriminalistika
9789531413695	Dnevnik Pauline P.	0004	004	2019	za mlade
9789531413480	Gorski dnevnik Pauline P.	0004	004	2012	za mlade
0000140620583	Hamlet	0005	005	1994	drama
0009530601905	Romeo i Julija	0005	006	2003	tragedija

Tablica 1.1: Relacija `Knjige`

ID_autor	Ime	Prezime
0001	Robert	Manger
0002	John R. R.	Tolkien
0003	Arthur C.	Doyle
0004	Sanja	Polak
0005	William	Shakespeare

Tablica 1.2: Relacija `Autori`

ID_izdavac	Ime
001	Element
002	Algoritam
003	Zagrebacka naklada
004	Mozaik knjiga
005	Penguin books
006	Skolska knjiga

Tablica 1.3: Relacija `Izdavaci`

1.2 Relacijska algebra

Relacijska algebra posebna je vrsta algebre i jezika za postavljanje upita, čiji je teorijski značaj važniji od praktičnog. Jezik se temelji na domeni sastavljenoj od relacija i operatora koji djeluju na relacijama. Izrazi u jeziku grade se induktivno koristeći operatore nad relacijama, a predstavljaju upite o bazi. Rezultat upita su ponovo relacije koje zadovoljavaju upit odnosno odgovaraju na njega. Važno svojstvo relacijske algebre, posebno vidljivo na primjerima korištenja operatora, je proceduralnost jezika. *Proceduralnost* znači da se izrazi definiraju „algoritamski", odnosno prateći zadani poredak operatora i operanada moguće je konstruirati plan izvršavanja upita do željenog rezultata. Također, poredak operatora i operanada u upitu možemo prikazati *sintaksnim stablom*, a plan izvršavanja će tada biti dan *inorder* obilaskom stabla.

Najvažniji dio jezika su operatori. Postoje 4 vrste operatora:

1. skupovni operatori,
2. operatori koji uklanjaju dijelove relacije,
3. operatori koji kombiniraju *n*-torke više relacija te
4. operator preimenovanja.

U nastavku ćemo definirati najvažnije operatore i prikazati njihovo djelovanje koristeći primjer 1.1.1. Također, za svaki ćemo operator navesti kojoj vrsti pripada[2].

Definicija 1.2.1. *Za relacije R i S kažemo da su* ***kompatibilne*** *ako imaju jednake sheme i domene svakog od atributa te su atributi jednako poredani u obje relacije.*

Definicija 1.2.2. *Za kompatibilne relacije R i S definiramo:*

- ***uniju****, u oznaci* $R \cup S$*, kao skup n-torki koje se nalaze u R ili u S (moguće i u obje),*
- ***presjek****, u oznaci* $R \cap S$*, kao skup n-torki koje se nalaze i u R i u S,*
- ***(skupovnu) razliku****, u oznaci* $R \setminus S$*, kao skup n-torki koje se nalaze u R, ali se ne nalaze u S.*

Unija, presjek i skupovna razlika pripadaju skupovnim operatorima.

[2] Češće ćemo reći da operator *pripada* nekoj skupini (ili vrsti) nego da *je* neke vrste. Dodatno, definirani operatori neće biti jedini predstavnici navedenih skupina.

Primjer 1.2.3. *Relacije* `Autori` *i* `Izdavaci` *iz primjera 1.1.1 nisu kompatibilne jer nemaju jednaki broj atributa. Relacija* `Izdavacke_kuce` *sa shemom:* `Izdavacke_kuce(ID_izdavac, Ime)` *je kompatibilna s relacijom* `Izdavaci`.

Djelovanje skupovnih operatora nećemo prikazivati na primjeru jer je njihovo djelovanje identično djelovanju nad skupovima i iz definicije se lako vidi koja je rezultantna relacija tih operacija.

Relacijska algebra omogućuje nam mijenjanje izgleda i broja n-torki u relaciji. Uklanjanjem n-torki iz relacije koristeći operator selekcije utječemo na broj, dok uklanjanjem atributa unutar relacije operatorom projekcije utječemo na izgled n-torki. Stoga, projekcija i selekcija pripadaju operatorima koji uklanjanju dijelove relacije.

Definicija 1.2.4. ***Projekcija*** *je unarni operator čijim djelovanjem dobivamo relaciju koja se sastoji od podskupa skupa atributa relacije R, a taj zadani podskup nazivamo* ***projekcijska lista****. Djelovanje operatora opisano je izrazom:* $\pi_{\text{projekcijska lista}}(R)$.

Selekcija *je unarni operator čijim djelovanjem dobivamo relaciju sastavljenu od n-torki relacije R koje zadovoljavaju zadani uvjet, a taj uvjet nazivamo* ***selekcijski uvjet****. Djelovanje operatora je opisano izrazom:* $\sigma_{\text{selekcijski uvjet}}(R)$. *Uvjeti kod selekcije grade se od logičkih veznika (*I, ILI, NE*), operatora usporedbe (*$<, >, \leqslant, \geqslant, \neq, =$*) i atributa ili konstanti.*

Napomena 1.2.5. *Prilikom definiranja pojma relacije istaknuli smo važno svojstvo jedinstvenosti n-torki unutar relacije, stoga je važno napomenuti kako se u relaciji dobivenoj projekcijom brišu duplikati (identične n-torke) te je promijenjena shema polazne relacije. U rezultantnoj relaciji selekcije nema duplikata, a shema je jednaka kao u polaznoj relaciji.*

Primjer 1.2.6. *Prikažimo djelovanje operatora selekcije i projekcije koristeći primjer 1.1.1. Konstruirat ćemo upit u relacijskoj algebri kojim želimo dobiti sve žanrove knjiga, a za to koristimo relaciju* `Knjige` *i operator projekcije. Rezultat je prikazan u tablici 1.4.*

Za primjer selekcije koristit ćemo relaciju `Knjige` *i konstruirat ćemo upit kojim želimo dobiti knjige izdane nakon 2010. godine. Operatori relacijske algebre mogu se međusobno komponirati. Kako bismo prikazali komponiranje selekcije i projekcije, želimo dohvatiti samo ime knjige i pripadnu godinu. Rezultat je prikazan u tablici 1.5, a upit glasi:*

$$\sigma_{\texttt{Godina} > 2010}(\pi_{\texttt{Ime},\texttt{Godina}}(\texttt{Knjige})) \tag{1.1}$$

U nastavku ćemo definirati operatore koji kombiniraju više relacija. Prilikom kombiniranja n-torki iz više relacija može se dogoditi da neki od atributa u rezultantnoj relaciji imaju isto ime. Kako ne bismo imali istoimene stupce u relaciji, koristimo najjednostavniji operator relacijske algebre, **operator preimenovanja**. Neka je R relacija stupnja n te neka su $A_1, \ldots, A_n$ imena atributa. Djelovanje operatora dano je izrazom: $\rho_{S(A_1, \ldots, A_n)}(R)$. Nova

Zanr
obrazovanje
fantastika
kriminalistika
za mlade
drama
tragedija

Tablica 1.4: Rezultat upita $\pi_{\texttt{Zanr}}(\texttt{Knjige})$

Ime	Godina
Baze podataka	2014
Softversko inzenjerstvo	2016
Crveni krug	2015
Dnevnik Pauline P.	2019
Gorski dnevnik Pauline P.	2012

Tablica 1.5: Relacija dobivena upitom 1.1

relacija imena S je nastala izvršavanjem prethodnog izraza, a njezine n-torke su jednake onima u relaciji R, osim što su imena atributa redom imena $A_1, \ldots, A_n$.

Definicija 1.2.7. *Neka je R relacija stupnja n i S relacija stupnja m. Operator* ***Kartezijevog produkta*** *nad R i S je relacija $R \times S$ sastavljena od svih $(n + m)$-torki koje su nastale spajanjem proizvoljne n-torke iz R s proizvoljnom m-torkom iz S. Broj $(n + m)$-torki u relaciji $R \times S$ je $n \cdot m$.*

Primjer 1.2.8. *Kombiniranje n-torki s m-torkama iz definicije 1.2.7 lakše se vidi na primjeru. Za ovaj ćemo primjer dodati u bazu relaciju* `Suradnja` *sljedeće relacijske sheme:* `Suradnja(ID_autor, ID_izdavac)`*. Relacija je prikazana u tablici 1.6. Identifikacijske oznake autora i izdavača nalaze se u relaciji ako je izdavačka kuća objavila barem jedno djelo autora.*

Ako nas zanima s kojim izdavačkim kućama autori nisu surađivali, napisat ćemo sljedeći upit koristeći Kartezijev produkt i skupovnu razliku:

$$\left(\pi_{\texttt{ID\_autor}}(\texttt{Autori}) \times \pi_{\texttt{ID\_izdavac}}(\texttt{Izdavaci})\right) \setminus \texttt{Suradnja}\,. \tag{1.2}$$

Kako bismo dobili traženo, najprije smo napravili sve moguće kombinacije identifikacijskih oznaka autora i izdavača Kartezijevim produktom, a potom iz relacije skupovnom razlikom uklonili sve one parove koji su surađivali. Rezultat vidimo u tablici 1.7.

Osim što retke dviju tablica možemo kombinirati Kartezijevim produktom, postoji skupina takozvanih *join* operatora, **operatora spajanja**, koji spajaju dvije relacije na zadani način, primjerice po nekom zajedničkom atributu, nekom uvjetu zapisanom pomoću operatora usporedbe i slično. Primijetimo da ponovo postoji mogućnost pojave istoimenih atributa u rezultantnoj relaciji, no to rješavamo koristeći operator preimenovanja. U nastavku ćemo definirati prirodni spoj i theta-spoj. Napominjemo da u operatore spajanja pripadaju još i vanjski spoj [12, str. 219] te spajanje po jednakosti (eng. *equijoin*) [13].

ID_autor	ID_izdavac
0001	001
0002	002
0002	004
0002	005
0002	006
0003	003
0003	005
0003	006
0004	003
0004	004
0005	002
0005	004
0005	005
0005	006

Tablica 1.6: Relacija `Suradnja`

ID_autor	ID_izdavac
0001	002
0001	003
0001	004
0001	005
0001	006
0002	001
0002	003
0003	001
0003	002
0003	004
0004	001
0004	002
0004	005
0004	006
0005	001
0005	003

Tablica 1.7: Rezultat upita 1.2

Definicija 1.2.9. *Neka su R i S relacije. Definiramo* ***uvjet spoja*** *C kao konjunkciju jednog ili više izraza oblika A* op *B, gdje je A neki atribut relacije R i B neki atribut relacije S, a* op *je jedan od operatora usporedbe* ($<, >, \leqslant, \geqslant, \neq, =$).

Operator ***theta-spoja*** *relacija R i S je skup n-torki Kartezijevog produkta $R \times S$ koje zadovoljavaju uvjet spoja C. Djelovanje operatora dano je izrazom: $R \bowtie_C S$.*

Napomena 1.2.10. *Theta-spoj je izvedeni operator; može se dobiti selekcijom i Kartezijevim produktom: $\sigma_C (R \times S)$.*

Primjer 1.2.11. *Za potrebe prikaza djelovanja theta-spoja dodat ćemo dvije nove relacije. Prva relacija* `Cjenik` *bit će vezana uz cijene knjiga (tablica 1.10). Druga relacija je* `Kategorija`*, sheme* `Kategorija(Min, Max, Ime)`*, vidljiva u tablici 1.8. Atribut* `Ime` *označava ime kategorije kojoj knjiga može pripadati u ovisnosti o cjenovnom rangu; minimalna vrijednost dana je s* `Min`*, maksimalna s* `Max`*.*

Upitom želimo saznati kojoj kategoriji pripada svaka knjiga u ovisnosti o cijeni. Upit koristi theta-spoj kako bismo dobili jedinstvenu kategoriju za svaku knjigu, što možemo vidjeti u tablici 1.9. Upit izgleda ovako:

$$\pi_{ISBN,\,Ime}\,(Cjenik \bowtie_{Cijena\_kn \leqslant Max\ I\ Min \leqslant Cijena\_kn}\ Kategorija)\,. \tag{1.3}$$

Min	Max	Ime
0	99	jeftino
100	189	pristupacno
190	1000	skupo

Tablica 1.8: Relacija `Kategorija`

ISBN	Ime
9789531975766	pristupacno
9789531976008	pristupacno
0009536166119	skupo
0009536166118	skupo
0009536166135	skupo
9789532521849	jeftino
9789531413695	jeftino
9789531413480	jeftino
0000140620583	pristupacno
0009530601905	pristupacno

Tablica 1.9: Rezultat upita 1.3

Definicija 1.2.12. *Neka su R i S relacije, stupnja redom n i m te neka R i S imaju l atributa koji se nalaze i u R i u S, gdje je* $l \in \mathbb{N}$, $l \leqslant n$ *i* $l \leqslant m$. ***Prirodni spoj*** *relacija R i S, u oznaci* $R \bowtie S$*, je skup* $(n + m - l)$*-torki dobiven spajanjem n-torke iz R s m-torkom iz S koje zadovoljavaju svojstvo da za svaki atribut koji se nalazi i u R i u S vrijedi da su im vrijednosti tog atributa jednake.*

Napomena 1.2.13. *Rezultantna relacija prirodnog spoja ima drugačiju shemu od Kartezijevog produkta R i S; zajednički atributi relacija R i S se pojavljuju samo jednom.*

Primjer 1.2.14. *U ovom primjeru ćemo koristiti relaciju* `Cjenik`*, vidljivu u tablici 1.10. Napisat ćemo upit kojim ćemo dobiti imena knjiga koje su na popustu. U tablici 1.11 možemo vidjeti rezultat upita:*

$$\pi_{\texttt{Ime},\texttt{Cijena\_kn}}\left(\sigma_{\texttt{Popust}=\text{"da"}}(\texttt{Cjenik}) \bowtie \texttt{Knjige}\right). \tag{1.4}$$

U literaturi se mogu pronaći i drugi operatori relacijske algebre, no mi ih u ovome radu nećemo definirati jer se rjeđe koriste ili se mogu izvesti koristeći opisane operatore. Primjeri takvih operatora su dijeljenja relacija [13] i već spomenuti vanjski spoj.

Nadopunjavanjem relacijske algebre novim operatorima dobivamo razna proširenja, poput proširenja s agregatnim funkcijama, proširenja za rad s multiskupovima [6] i slično. Jednim primjerom proširenja relacijske algebre bavit ćemo se u točki 3.2.

Uz relacijsku algebru, često se spominje deklarativni jezik za postavljanje upita, istog autora Edgara Codda, pod nazivom **relacijski račun**. *Deklarativni* znači da upit definira koje to *n*-torke želimo dobiti, no ne govori ništa o tome kako ih dobiti, za razliku od proceduralne relacijske algebre gdje smo mogli graditi upit tako da konstruiramo izvršavanje

ISBN	Cijena_kn	Popust
9789531975766	110	ne
9789531976008	144	ne
0009536166119	199	da
0009536166118	199	da
0009536166135	199	da
9789532521849	50	da
9789531413695	79	ne
9789531413480	79	ne
0000140620583	129	ne
0009530601905	129	da

Tablica 1.10: Relacija `Cjenik`

Ime	Cijena_kn
Prstenova druzina	199
Dvije kule	199
Povratak kralja	199
Crveni krug	50
Romeo i Julija	129

Tablica 1.11: Rezultat upita 1.4

upita. U upitu deklarativnog jezika prepoznajemo predikat[3] koji n-torke moraju zadovoljavati da bi bile u rezultantnoj relaciji. Razlog zašto spominjemo postojanje relacijskog računa je njegov utjecaj na stvaranje jezika SQL, koji je tema idućeg poglavlja.

[3] Predikat P mjesnosti k definira se kao podskup Kartezijevog produkta k skupova i za k-torku $\vec{x}$ vrijedi $\vec{x} \in P$ ako i samo ako ona zadovoljava predikat, odnosno ako vrijedi $P(\vec{x})$.

Poglavlje 2

SQL

Najpoznatiji i najraširenij jezik za rad s relacijskim bazama podataka je SQL, što je kratica od *Structured Query Language*. U uvodu smo opisali SQL kao integrirani jezik za rad s bazama. Izrazima u SQL-u moguće je definirati novu tablicu, unositi podatke, modificirati postojeće podatke, vršiti izračune i upite, upravljati sigurnošću i ovlaštenjima u bazi i još mnogo toga. Nas će u ovome radu zanimati samo dio SQL-a vezan uz upite.

U ovome ćemo poglavlju navesti svojstva SQL-a kao jezika za postavljanje upita. U nastavku ćemo opisati operatore i koristeći primjer iz prethodnog poglavlja prikazati njihovo korištenje. Opisani pregled SQL-a bit će zapravo njegova jezgrena verzija, koja pokriva standard SQL-a iz 1992. godine. Na kraju poglavlja navest ćemo kratku povijest SQL-a, vezanu uz verzije SQL-a i nove elemente koji su uvedeni u jezik. Posebnu ćemo pažnju posvetiti elementu nadogradnje jezika — rekurziji. Definirat ćemo ju, navesti svojstva i na primjerima pokazati važnost i korisnost njezinog uvođenja u jezik.

2.1 SQL kao jezik za postavljanje upita

Relacijska algebra, kao i relacijski račun, utjecali su na stvaranje SQL-a. Deklarativnost jezika SQL je ono što ga čini sličnijim relacijskom računu. U nastavku rada ćemo uspoređivati izražajnu moć SQL-a i relacijske algebre, uzimajući u obzir Coddov teorem, da relacijska algebra i račun imaju istu izražajnu moć [21].

Iako je SQL također baziran na relacijskom modelu, velika razlika u odnosu na relacijsku algebru je rad s multiskupovima n-torki. Konkretno, SQL relacije mogu imati istovjetne n-torke u relaciji za razliku od relacijske algebre; u literaturi se takva relacija naziva **vreća** (eng. *bag*). Još jedna razlika, vezana uz relacije, je i u imenovanju stupaca. Naime, SQL dopušta jednaka imena stupaca prilikom spajanja relacija. Primjerice, neka su R i S proizvoljne relacije, neka i R i S sadrže stupac A; tada će SQL stupce označavati kao $R.A$ i $S.A$.

Prije nego krenemo s opisima dijelova osnovnog upita, važno je napomenuti kako je rezultat izvođenja upita relacija koja je privremena i služi za ispis na ekranu ili prosljeđivanje nekoj aplikaciji. Također, upite možemo međusobno komponirati, odnosno jedan se upit može pojaviti kao podupit u drugome. Iako je standardom iz 1992. godine obavezno završavati upit znakom ';', u literaturi se on često izostavlja. Razlog tome je što ';' služi za odvajanje naredbi, a u literaturi rijetko promatramo više uzastopnih naredbi.

Osnovni oblik upita

Osnovni oblik izraza ili upita jezika SQL glasi:

$$\begin{aligned}&\texttt{SELECT } A_1, \ldots, A_n\\&\texttt{FROM } R_1, \ldots, R_m\\&\texttt{WHERE } C\ .\end{aligned}$$

Klauzule SELECT i FROM čine jezgru svakog upita. Rezultat izvođenja je relacija sastavljena od Kartezijevog produkta relacija $R_1, \ldots, R_m$ navedenih nakon klauzule FROM. Stupci te relacije su $A_1, \ldots, A_n$ navedeni nakon klauzule SELECT. Nakon SELECT, umjesto liste atributa, moguće je napisati znak '$*$'. Tada će u rezultantnoj relaciji biti uključeni svi stupci željenih relacija. Uključimo li klauzulu WHERE i dodamo uvjet C, rezultat upita će biti samo one n-torke koje zadovoljavaju uvjet C. Unutar uvjeta C mogu se pojaviti ugniježđeni upiti (npr. upit 2.4). Detalji o izgledu uvjeta bit će izneseni u sljedećem odjeljku.

Primjer 2.1.1. *Koristeći primjer 1.1.1 prikazat ćemo korištenje osnovnog oblika upita. Želimo dobiti imena i prezimena svih autora iz tablice* Autori*. Tablica 2.1 predstavlja rezultat odgovarajućeg upita:*

$$\textit{SELECT Ime, Prezime FROM Autori}\ . \tag{2.1}$$

Upit je mogao izgledati i ovako: $\textit{SELECT} * \textit{FROM Autori}$*. Rezultat bi tada bio kao u tablici 1.2.*

Ime	Prezime
Robert	Manger
John R. R.	Tolkien
Arthur C.	Doyle
Sanja	Polak
William	Shakespeare

Tablica 2.1: Rezultat upita 2.1

Sljedećim upitom želimo dobiti imena knjiga koje su izdane prije 2010. godine zajedno s imenom i prezimenom autora. Rezultat izvođenja je prikazan u tablici 2.2, a upit glasi:

```
SELECT Autori.Ime, Prezime, Knjige.Ime
FROM Autori, Knjige                                    (2.2)
WHERE Autori.ID_autor = Knjige.ID_autor AND Godina < 2010 .
```

Autori.Ime	Prezime	Knjige.Ime
John R. R.	Tolkien	Prstenova druzina
John R. R.	Tolkien	Dvije kule
John R. R.	Tolkien	Povratak kralja
William	Shakespeare	Hamlet
William	Shakespeare	Romeo i Julija

Tablica 2.2: Rezultat izvođenja upita 2.2

Primijetimo kako je poredak stupaca u rezultantnoj tablici identičan onom navedenom nakon klauzule `SELECT` i kako su imena stupaca proširena imenom relacije iz koje dolaze. Štoviše, iz Kartezijevog produkta dviju zadanih tablica izabrali smo one n-torke koji imaju isti ID autora i čija je godina izdavanja manja od 2010.

Napomena 2.1.2. *Prokomentirat ćemo na koji bi način `SELECT...FROM...WHERE...` upit odgovarao upitu relacijske algebre. S obzirom na opisano djelovanje, možemo zaključiti da će `SELECT` odgovarati operatoru projekcije π, dok će operator selekcije σ odgovarati klauzuli `WHERE`. Također, u `FROM` dijelu upita formira se Kartezijev produkt zadanih tablica, a taj smo operator definirali i u relacijskoj algebri. Osnovni oblik upita u SQL-u može se zapisati sljedećim izrazom relacijske algebre: $\pi_{A_1,\ldots,A_n}(\sigma_C(R_1 \times \cdots \times R_m))$.*

Izgled `WHERE` uvjeta

U primjeru 2.1.1 vidjeli smo jedan način definiranja uvjeta nakon klauzule `WHERE`. Uvjet se gradi od logičkih veznika (`AND`, `NOT`, `OR`) i operatora usporedbe, na sličan način kao u relacijskoj algebri. Operatori usporedbe u SQL-u su: =, <=, >=, <, >, <>[1]. Također, unutar uvjeta mogu se pojaviti i aritmetički operatori (npr. +, -, *, /). Primjerice, uvjet može izgledati ovako: `(Godina - 2010) < 10`.

Osim navedenog, u uvjetu se mogu pojaviti dodatne ključne riječi. Svaka od njih služi kako bismo odabrali retke od interesa. Ključnu riječ `LIKE` koristimo za tzv. *pattern matching*, kao što ćemo vidjeti u primjeru 2.1.3. Ključnu riječ `BETWEEN` koristimo kada želimo

[1] Za vrijednosti `a` i `b`, vrijedi `a <> b` ako su `a` i `b` međusobno različiti.

provjeriti nalaze li se neke vrijednosti unutar nekog raspona. Izraz u uvjetu je oblika `A BETWEEN` v_1 `AND` v_2, a provjeravamo nalaze li se vrijednosti stupca `A` unutar raspona od v_1 do v_2.

Sljedeća ključna riječ koja se može pojaviti je `IN`. Ona se koristi u složenijim upitima najčešće uz ugniježđene upite, a može se koristiti i s negacijom, `NOT IN`. Korištenjem izraza oblika `A IN R` unutar `WHERE` uvjeta želimo dobiti sve redove gdje se vrijednost stupca `A` nalazi u `R`. Relacija `R` tada mora biti unarna, odnosno mora imati samo jedan stupac. Dodatno, `R` možemo zadati i kao skup vrijednosti $v_1, v_2, \ldots, v_n$, za $n \in \mathbb{N}$. Tada bi upit izgledao ovako: `A IN (`$v_1, v_2, \ldots, v_n$`)`.

`EXISTS` je još jedna ključna riječ koja se koristi u složenijim upitima. Izrazom oblika `EXISTS R` provjeravamo postojanje redova u tablici `R` (koja može biti rezultat ugniježđenog upita), analogno radi i `NOT EXISTS`. Ključne riječi koje se koriste na sličan način kao `IN` i `EXISTS` su `ANY` i `ALL`. One se koriste uz ime stupca i nekog operatora usporedbe, konkretno u obliku: `A` op `ANY R` ili `A` op `ALL R`. Relacija `R` tada mora biti unarna, a vrijednosti u stupcu relacije `R`, kao i vrijednosti u stupcu `A`, moraju se moći uspoređivati operatorom op. Praktično korištenje navedenih ključnih riječi prikazat ćemo u primjeru 2.1.3.

Primjer 2.1.3. *U ovome ćemo primjeru pokazati neke oblike uvjeta nakon klauzule* `WHERE` *i korištenje nekih ključnih riječi. Želimo dobiti popis izdavača u čijem se imenu nalazi riječ* `'knjiga'`*. Rezultat vidimo u tablici 2.3, a upit glasi:*

```
SELECT Ime FROM Izdavaci WHERE Ime LIKE '%knjiga%' .        (2.3)
```

Koristit ćemo ključnu riječ `EXISTS`*, odnosno* `NOT EXISTS`*, kako bismo dobili sve autore koji nisu surađivali s izdavačem identifikacijske oznake 004, konkretno Mozaik knjigom. U tu ćemo svrhu koristiti tablicu 1.6 (*`Suradnja`*) iz primjera 1.2.8. Upit izgleda ovako:*

```
SELECT Ime, Prezime FROM Autori WHERE NOT EXISTS
(SELECT * FROM Suradnja                                     (2.4)
WHERE Suradnja.ID_autor = Autori.ID_autor AND ID_izdavac = 004) ,
```

a rezultat je dan u tablici 2.4.

Nadalje, želimo dobiti imena knjiga koje nisu na popustu. Ovdje ćemo prikazati korištenje ugniježđenih upita s ključnom riječi `IN` *u kombinaciji s logičkim operatorom* `NOT`*. Koristit ćemo tablicu 1.10 (*`Cjenik`*) iz primjera 1.2.14. Upit glasi:*

```
SELECT Ime FROM Knjige
WHERE ISBN NOT IN                                           (2.5)
(SELECT ISBN FROM Cjenik WHERE Popust = 'da') ,
```

a rezultat je vidljiv u tablici 2.5.

Ime
Mozaik knjiga
Skolska knjiga

Tablica 2.3: Rezultat upita 2.3

Ime	Prezime
Robert	Manger
Arthur C.	Doyle

Tablica 2.4: Rezultat upita 2.4

Prikazat ćemo korištenje ključne riječi `ALL` *u upitu kojim želimo dobiti imena svih knjiga koje su izdane nakon svih obrazovnih knjiga. Rezultat je vidljiv u tablici 2.6, a odgovarajući upit koji koristi* `ALL` *izgleda ovako:*

```
SELECT Ime FROM Knjige
WHERE Godina > ALL                                              (2.6)
(SELECT Godina FROM Knjige WHERE Zanr = 'obrazovanje') .
```

Korištenje ključne riječi `ANY` *analogno je korištenju ključne riječi* `ALL`*.*

Ime
Baze podataka
Softversko inzenjerstvo
Dnevnik Pauline P.
Gorski dnevnik Pauline P.
Hamlet

Tablica 2.5: Rezultat upita 2.5

Ime
Dnevnik Pauline P.

Tablica 2.6: Rezultat upita 2.6

Skupovne operacije

Među mogućnostima SQL-a naveli smo rad s multiskupom n-torki. Ako se želimo riješiti duplikata među rezultantnim redovima, dovoljno je koristiti ključnu riječ `DISTINCT`. Ona se najčešće pojavljuje između ključnih riječi `SELECT` i `FROM`. Nadalje, imamo jednostavan, ali koristan operator preimenovanja (slično kao u relacijskoj algebri). Izrazom oblika `SELECT A AS B` stupac `A` preimenovali smo u `B` u rezultantnoj relaciji. Analogno, izrazom oblika `FROM R AS S` preimenujemo relaciju `R` u `S`. Napominjemo da preimenovanje nije trajno; relacija `R` neće promijeniti ime u `S` u bazi. Preimenovanje na ovaj način omogućava da privremeno u upitu relaciju `R` „zovemo" `S`.

Skupovne operacije unije, presjeka i razlike vrše se nad multiskupovima u SQL-u i ponašaju se jednako kao u relacijskoj algebri. Općeniti upiti koji koriste uniju, presjek i

(skupovnu) razliku izgledaju ovako:

```
(SELECT ... FROM ... WHERE ...) UNION (SELECT ... FROM ... WHERE ...),
(SELECT ... FROM ... WHERE ...) INTERSECT (SELECT ... FROM ... WHERE ...),
(SELECT ... FROM ... WHERE ...) EXCEPT (SELECT ... FROM ... WHERE ...).
```

Spojevi

SQL omogućuje konstrukciju različitih vrsta spoja (eng. *join*). Ukratko ćemo ih ovdje navesti. Sinonim Kartezijevog produkta iz relacijske algebre u SQL-u je operator `CROSS JOIN` koji se poziva izrazom `R CROSS JOIN S`. Operator sličan theta-spoju je `JOIN`. Koristi se uz ključnu riječ `ON` (u obliku `R JOIN S ON C`), a rezultat upita je podskup Kartezijevog produkta relacija `R` i `S` koji sadrži točno one redove koji zadovoljavaju uvjet `C`. Operator prirodnog spoja pozivamo s `NATURAL JOIN`, a djeluje isto kao i njegov istoimeni operator u relacijskoj algebri. Analogno, vanjski spoj dobiva se s `OUTER JOIN`. Uz opisane, postoje razne podvrste spojeva, poput lijevog vanjskog spoja (`LEFT OUTER JOIN`).

Grupiranje i sortiranje

U SQL-u postoji način da n-torke particioniramo u manje grupe koje zadovoljavaju neke uvjete. Grupe se mogu koristiti kao relacije u složenijim upitima. U tu svrhu koristimo klauzulu `GROUP BY` koja slijedi nakon `WHERE` uvjeta. Redci relacije bit će grupirani u ovisnosti o vrijednostima atributa navedenih nakon klauzule `GROUP BY`. Djelovanje operatora bit će veoma korisno prilikom kombiniranja s agregatnim funkcijama, što ćemo vidjeti u primjeru 2.1.5.

Uz operaciju grupiranja često se veže klauzula `HAVING`. Njezino pozivanje i djelovanje je slično klauzuli `WHERE`. Naime, izgled upita koji koristi tu klauzulu izgleda kao:

```
SELECT ... FROM ... WHERE ... GROUP BY ... HAVING C .
```

Uvjet `C` djeluje nakon grupiranja i služi za odbacivanje grupa koje ne zadovoljavaju uvjet `C`. Primjenu ćemo demonstrirati u primjeru 2.1.5.

Spomenimo važnu operaciju sortiranja u SQL-u. Sortiranje se provodi koristeći ključnu riječ `ORDER BY` nakon koje slijedi lista stupaca po kojima želimo sortirati. Ključne riječi `DESC` i `ASC` mogu se pojaviti nakon svakog navedenog stupca za sortiranje, kako bismo definirali želimo li vrijednosti sortirati silazno ili uzlazno (pogledati primjer 2.1.4). Dodatno, moguće je koristiti `LIMIT broj` kako bismo uzeli samo određeni broj redova tablice.

Primjer 2.1.4. *U ovom ćemo primjeru pokazati kako koristiti sortiranje u SQL-u. Zadatak je sortirati knjige iz tablice 1.1 po godini izdavanja, od najranije izdane knjige do najnovije. U slučaju da knjige imaju istu godinu izdavanja, želimo ih sortirati silazno po imenu.*

Tablica 2.7 prikazuje rezultat upita kojim ćemo to dobiti, a glasi ovako:

```
SELECT Ime, Godina FROM Knjige ORDER BY Godina ASC, Ime DESC .
```
(2.7)

Ime	Godina
Hamlet	1994
Prstenova druzina	2002
Povratak kralja	2002
Dvije kule	2002
Romeo i Julija	2003
Gorski dnevnik Pauline P.	2012
Baze podataka	2014
Crveni krug	2015
Softversko inzenjerstvo	2016
Dnevnik Pauline P.	2019

Tablica 2.7: Rezultat upita 2.7

`NULL` vrijednost

Osvrnut ćemo se kratko na postojanje posebne vrijednosti koja se može nalaziti u stupcima tablica. Ukoliko je neki podatak nepoznat ili pogrešan, na njegovo je mjesto u tablici moguće upisati posebnu vrijednost — `NULL`. Djelovanje operatora jezika SQL, posebice logičkih operatora, u tom je slučaju posebno određeno. U literaturi, npr. [12, točka 6.1.6 i 6.1.7], se često logika SQL-a naziva **trovrijednosna logika** s obzirom na to da osim `TRUE` i `FALSE` kao vrijednosti logičkih izraza, postoji i `UNKNOWN` (vrijednost pridružena `NULL`).

`NULL` vrijednost ima utjecaja i na djelovanje posebnih vrsta funkcija — agregata koje ćemo opisati u sljedećoj točki. Jednostavan primjer korištenja `NULL` vrijednosti u upitima, kao i utjecaj na agregatne funkcije, vidjet ćemo u primjeru 2.1.6.

Agregatne funkcije

Postojanje agregatnih funkcija u SQL-u važna je razlika u odnosu na relacijsku algebru. Njihovo je korištenje vrlo učestalo u praktičnom korištenju jezika SQL. Agregatne funkcije pozivaju se nad jednim stupcem, a kao rezultat najčešće daju broj. Njihovo se korištenje često kombinira s operatorom grupiranja jer su nam katkad potrebne vrijednosti koje odgovaraju grupama n-torki. Proučit ćemo 5 agregatnih funkcija koje se najčešće koriste u SQL-u, a to su:

- `AVG(A)` - vraća srednju vrijednost vrijednosti stupca A koje su različite od `NULL`, ako takvih nema ili je stupac prazan rezultat je `NULL` vrijednost,
- `COUNT(A)` - vraća broj vrijednosti u stupcu A koje nisu `NULL`, ako takvih nema ili je stupac prazan vraća 0,
- `MIN(A)` - vraća minimalnu vrijednost u stupcu A koja nije `NULL`, ako takve nema ili je stupac prazan rezultat je `NULL` vrijednost,
- `MAX(A)` - vraća maksimalnu vrijednost u stupcu A koja nije `NULL`, ako takve nema ili je stupac prazan rezultat je `NULL` vrijednost,
- `SUM(A)` - vraća zbroj vrijednosti u stupcu A koje nisu `NULL`, ako takvih nema ili je stupac prazan rezultat je `NULL` vrijednost.

U primjeru 2.1.5 ćemo prikazati djelovanje ovih agregatnih funkcija, kao i kombiniranje s operatorom grupiranja. Agregatne funkcije koristimo nakon klauzule `SELECT` ili kao dijelove uvjeta u `WHERE` ili `HAVING` klauzuli. Unutar zagrada u `SUM`, `AVG`, `COUNT` funkcijama se može pojaviti ključna riječ `DISTINCT`; tada u izračun neće biti uključeni duplikati.

Nadalje, djelovanje agregatne funkcije `COUNT` uvelike će ovisiti o načinu pozivanja, ali i postojanju `NULL` vrijednosti u tablici. Funkciju `COUNT` možemo koristiti u obliku `COUNT(*)`, što znači da se ne mora nužno pozvati nad jednim stupcem. Pozovemo li `COUNT(*)` kao rezultat ćemo dobiti broj svih redaka u relaciji (ili grupi), neovisno o tome jesu li vrijednosti `NULL`. Zanimljivo je [9] da možemo napisati bilo koji izraz u funkciji `COUNT` i ponovo će rezultat biti kao i ranije. Dozvoljeni su izrazi poput `COUNT(1)`, `COUNT(-10)`, `COUNT('bok')`, no jedino nije dozvoljeno `COUNT()`. Međutim, kada kao argument napišemo ime stupca, onda `COUNT(A)`, gdje je `A` ime stupca, vraća broj vrijednosti u stupcu `A` koje nisu `NULL`.

Primjer 2.1.5. *Slijedi nekoliko upita u kojima želimo prikazati djelovanje operatora grupiranja i agregatnih funkcija. U tu ćemo svrhu ponovo koristiti tablice iz primjera 1.1.1, kao i dodatne tablice* `Cjenik` *(tablica 1.10 iz primjera 1.2.14) i* `Suradnja` *(tablica 1.6 iz primjera 1.2.8).*

Koristeći tablicu s cijenama knjiga želimo pronaći najjeftiniju i najskuplju knjigu, točnije njezinu cijenu, kao i srednju vrijednost cijena svih knjiga. Vrijednosti su vidljive u tablici 2.8, a jednostavan upit kojim izračunavamo željeno glasi:

```
SELECT MIN(Cijena_kn), MAX(Cijena_kn), AVG(Cijena_kn) FROM Cjenik.
```

Pokazat ćemo korištenje `GROUP BY` *i funkcije* `COUNT`*. Želimo dobiti broj knjiga određenog žanra u tablici 2.9 upitom:*

```
SELECT Zanr, COUNT(*) AS Broj FROM Knjige GROUP BY Zanr.        (2.8)
```

MIN(Cijena_kn)	MAX(Cijena_kn)	AVG(Cijena_kn)
50	199	131.7

Tablica 2.8: Minimalna, maksimalna i srednja cijena knjiga

Funkcija `COUNT` *pobrojat će retke koji odgovaraju istoj grupi, tj. u našem upitu retke koji imaju isti žanr.*

Preostalo nam je pokazati korištenje klauzule `HAVING`*. Želimo vratiti identifikacijsku oznaku autora koji je radio s više od dva izdavača. Ispisat ćemo ID autora i broj suradnji koje je imao. Zato ćemo koristiti tablicu* `Suradnja` *i funkciju* `COUNT` *u upitu oblika:*

```
SELECT ID_autor, COUNT(ID_autor) AS Suradnji
FROM Suradnja GROUP BY ID_autor                    (2.9)
HAVING COUNT(ID_autor) > 2 ,
```

rezultat je prikazan u tablici 2.10.

Primijetimo kako u oba slučaja `COUNT` *broji retke koji pripadaju nekoj grupi. Zbog nepostojanja* `NULL` *vrijednosti u tablicama na kojima se pozivaju upiti, rezultati upita 2.8 i 2.9 bili bi jednaki neovisno o načinu pozivanja (navodeći ime stupca ili znak '* $*$*').*

Zanr	Broj
obrazovanje	2
fantastika	3
kriminalistika	1
za mlade	2
drama	1
tragedija	1

Tablica 2.9: Rezultat upita 2.8

Id_autor	Suradnji
0002	4
0003	3
0005	4

Tablica 2.10: Rezultat upita 2.9

Primjer 2.1.6. *Za potrebe primjera dodat ćemo relaciju* `Narudzbe` *u koju pohranjujemo detalje narudžbi knjiga. Radi jednostavnosti reducirali smo atribute relacije (pogledati u tablici 2.11). Za neke od knjiga primjećujemo kako je vrijednost atributa* `Placeno` *jednaka* `NULL` *što će u našem primjeru značiti da knjiga nije plaćena.*

Upit kojim dobivamo sve knjige koje su plaćene izleda ovako:

```
SELECT ID, Placeno FROM Narudzbe WHERE Placeno IS NOT NULL,    (2.10)
```

a njihove identifikacijske oznake i pripadne datume plaćanja vidimo u tablici 2.12a.

ID	ISBN	Placeno
1	9789531975766	13/04/2022
2	9789531975766	NULL
3	0009536166135	NULL
4	0009536166135	12/06/2022
5	0009536166135	15/06/2022
6	0000140620583	NULL
7	0000140620583	NULL

Tablica 2.11: Tablica `Narudzbe`

ID	Placeno
1	13/04/2022
4	12/06/2022
5	15/06/2022

Tablica 2.12a: Rezultat upita 2.10

ISBN	COUNT(∗)
9789531975766	2
0009536166135	3
0000140620583	2

Tablica 2.12b: Rezultat upita 2.11

ISBN	COUNT(Placeno)
9789531975766	1
0009536166135	2
0000140620583	0

Tablica 2.12c: Rezultat upita 2.12

U sljedećim upitima ćemo uočiti razlike u izvršavanju agregatne funkcije `COUNT` *nad* `NULL` *vrijednostima u ovisnosti o načinu pozivanja. Neka prvi upit izgleda ovako:*

```
SELECT ISBN, COUNT(*) FROM Narudzbe GROUP BY ISBN,        (2.11)
```

a njime bismo zapravo dobili koliko je primjeraka svake od knjiga naručeno. Rezultat tog upita vidimo u tablici 2.12b te primijetimo razlike u odnosu na tablicu 2.12c. Ta je tablica rezultat upita oblika:

```
SELECT ISBN, COUNT(Placeno) FROM Narudzbe GROUP BY ISBN,  (2.12)
```

kojim dobivamo broj plaćenih primjeraka svake knjige.

Agregatne funkcije ćemo u ovome radu zapisivati kao u [16]. Naime, agregatna funkcija $\mathcal{F}$ je skup funkcija $\mathcal{F} = \{f_0, f_1, \ldots, f_\omega\}$, gdje je f_k funkcija koja uzima vreću[2] s k elemenata iz `Num` (oznaka za domenu s numeričkim vrijednostima) i vraća element iz `Num`. Funkciju nad praznom vrećom označavamo s f_0, a s f_ω funkciju nad beskonačno velikim vrećama. Kada bi `Num` označavao $\mathbb{N}, \mathbb{Q}$ ili $\mathbb{R}$, agregatne funkcije spomenute do sad definirale bi se na ovaj način:

[2]Umjesto { } zagrada kao kod skupova, za vreće koristimo {⟦ ⟧}.

- `SUM` kao $\sum = \{s_0, s_1, \dots\ s_\omega\}$ gdje je $s_k(\{\!\{x_1, \dots, x_k\}\!\}) = \sum_{i=1}^{k} x_i$, uz $s_0 = s_\omega = 0$,
- `COUNT` kao $C = \{c_0, c_1, \dots\ c_\omega\}$ gdje $c_k(\{\!\{x_1, \dots, x_k\}\!\})$ vraća k, uz $c_0 = c_\omega = 0$,
- `AVG` kao $\mathcal{A} = \{a_0, a_1, \dots, a_\omega\}$ gdje je $a_k(\{\!\{x_1, \dots, x_k\}\!\}) = \frac{s_k(\{\!\{x_1,\dots,x_k\}\!\})}{c_k(\{\!\{x_1,\dots,x_k\}\!\})}$, uz $a_0 = a_\omega = 0$,
- `MAX` kao $\mathcal{M} = \{m_0, m_1, \dots, m_\omega\}$ gdje je $m_k(\{\!\{x_1, \dots, x_k\}\!\}) = \max_i x_i$, uz $m_0 = m_\omega = 0$, analogno za `MIN` oznake $\mathfrak{m}$.

U ovim formalnim definicijama ponovo moramo pripaziti na pojavu `NULL` vrijednosti. Ako je vreća sastavljena samo od `NULL` vrijednosti smatra se da je vreća prazna, a inače se svaka `NULL` vrijednost u vreći zanemaruje. Ovakav formalni zapis agregatnih funkcija će nam trebati prilikom proširenja relacijske algebre agregatnim funkcijama u točki 3.2.

2.2 Verzije SQL-a

SQL je nastao oko 1974. pod imenom SEQUEL. Razvijen je u IBM-u, po radu Edgara Codda (autor relacijskog modela, algebre i računa), a tvorcima se smatraju D. D. Chamberlin i R. F. Boyce.

Prvi službeni standard koji se spominje u literaturi [12, poglavlje 6] je onaj iz 1986. godine, pod nazivom SQL-86. Jezik je tada standardizirao ANSI (kratica od *American National Standards Institute*). Međunarodna organizacija za standardizaciju (kratica ISO od eng. *International Organization for Standardization*) standard je usvojila 1987. godine. Svaku sljedeću standardizaciju usvojili su i ANSI i ISO; svaki od njih za novu standardizaciju objavljuje dokument pod nazivom u skladu sa svojim oznakama, ali su objavljeni standardi tih dviju organizacija sadržajno isti. O detaljima standardizacije možete pogledati u [22, dodatak F].

Pregled standarda (prema [15]) nastavljamo manjim ispravkom standarda iz 1986., što je rezultiralo standardom iz 1989. naziva SQL-89. Sljedeća važnija standardizacija dogodila se 1992. godine i naziva se SQL-92 (učestalije se spominje pod nazivom SQL2). Standardizacija iz 1999. pod nazivom SQL3 bit će u fokusu ovog rada. Naime, njome je uvedena rekurzija u jezik, uz još neke bitne elemente. Najnoviji standard objavljen je 2019.

Napominjemo još da SQL ima mnogo različitih dijalekata. Dijalekti su jezici koji su nastali na temelju SQL-a, ali svaki od njih je vezan uz svoj sustav za upravljanje bazom podataka i razvija se neovisno o SQL-u. Primjeri dijalekata su MySQL, PostgreSQL, T-SQL i tako dalje.

2.3 SQL3 i rekurzija

Standard iz 1999. godine, pod imenom SQL3, važan je u povijesti SQL-a zbog značajnog proširenja mogućnosti jezika, a u ovoj se točki bavimo njegovim najvažnijim novouvedenim elementom. SQL3 omogućuje sastavljanje rekurzivnih upita, a za to se koriste ključne riječi `WITH` i `RECURSIVE`. Iako je SQL jezik s relativno malim brojem ključnih riječi, s njima je moguće dobiti odgovor na brojne složene upite. Postavlja se pitanje zašto je uvođenje rekurzije toliko značajno za jezik. Je li razlog uvođenja jednostavniji zapis već mogućih upita ili omogućavanje sasvim novih vrsta upita?

Razlog uvođenja novih ključnih riječi u SQL3 standard je neiskazivost rekurzivnih upita [16] unutar okvira propisanih standardom SQL2 [1]. Proširivanjem jezika uvode se nove funkcionalnosti koje će omogućiti postavljanje rekurzivnih upita. Detaljima ograničenja SQL2 standarda u pogledu rekurzivnih upita bavit ćemo se u sljedećem poglavlju, dok ćemo u ovoj točki ukratko dati motivacijske primjere za uvođenje rekurzije, a potom njenu definiciju i primjene.

Rekurzivna funkcija je funkcija koja poziva samu sebe te pritom ima definiran bazni slučaj kojim se osigurava zaustavljanje njenog izračunavanja na intendiranoj domeni. Primjeri rekurzivnih funkcija su Fibonaccijev niz, faktorijele ili sume prvih nekoliko prirodnih brojeva. Navedeni su primjeri vezani uz matematiku, no rekurzija se ne pojavljuje samo u teorijskim razmatranjima. Praktični primjeri iz baza podataka koji se najčešće povezuju s rekurzijom su sljedeći:

1. Baza podataka u kojoj spremamo letove između gradova [12, točka 10.2.1]. Svaki redak tablice predstavlja jedan let iz početnog grada u završni. Upitom želimo dobiti sve gradove do kojih je moguće doći iz početnog grada s jednim ili više letova.

2. Baza podataka sadrži tablicu s roditeljima i njihovom djecom [20]. Svaki redak tablice predstavlja par roditelj-dijete. Potrebno je pronaći sve pretke određene osobe.

3. Baza podataka s financijskim podacima, poput stanja na računu, iznosa kredita, štednje, investicija i slično. Rekurzivni upiti se koriste za razne izračune i stvaranje izvješća.

Opći oblik rekurzije u SQL-u koristeći ključne riječi `WITH` i `RECURSIVE` izgleda ovako:

```
WITH RECURSIVE R AS
  (<bazni upit> UNION <rekurzivni upit>)
<upit s R> .
```

Primijetimo bazni slučaj rekurzije; u SQL upitu to je *<bazni upit>* u kojem se ne pojavljuje relacija `R`, dok se unutar *<rekurzivni upit>* pojavljuje relacija `R`. Napomenimo i

kako je moguće zadavanje rekurzivnog upita u obliku:

$$\texttt{WITH RECURSIVE R(A}_1\texttt{, A}_2\texttt{,..., A}_\texttt{m}\texttt{) AS ...,}$$

odnosno moguće je eksplicitno zadati imena atributa ($\texttt{A}_1$, $\texttt{A}_2$, ..., $\texttt{A}_\texttt{m}$) relacije R. Atributi relacije R zapravo se izvode iz <*bazni upit*> i <*rekurzivni upit*> pa se navođenjem imena na prikazani način mogu samo preimenovati atributi te relacije, a ne i dodavati novi atributi u relaciju.

Izvršavanje rekurzivnog upita razlikuje se od izvršavanja rekurzivne funkcije. Razlika je vezana uz redoslijed rekurzivnih poziva. Kod izvršavanja rekurzivnih funkcija se u skladu sa zadanim rekurzivnim izrazom u svakoj iteraciji rekurzije poziva rekurzivna funkcija s drugačijim ulazom, sve dok ne dođemo do baznog slučaja. On se prvi izračuna te se potom koristeći rezultate prošlih rekurzivnih poziva računaju rekurzivni izrazi dok ne dobijemo rezultat početnog rekurzivnog poziva.

Na početku izvršavanja rekurzivnog upita se izvrši bazni upit, nad čijim se rezultantnim n-torkama uz početne n-torke u sljedećoj iteraciji rekurzije poziva rekurzivni upit, i tako redom dalje dok ne dođemo do prazne rezultantne relacije[3]. Tada rekurzija staje i vraća se unija svih do tad izračunanih relacija.

Usporedbu izvršavanja rekurzivne funkcije i rekurzivnog upita prikazat ćemo na primjeru faktorijela. Neka je $f : \mathbb{N} \to \mathbb{N}$ funkcija koja računa faktorijel prirodnog broja n, definirana kao $f(n) = n \cdot f(n-1)$, za $n > 1$, uz $f(1) = 1$. Programski kod izgleda ovako:

```
def fact(n):
    if n<=1:
        return 1
    return n*fact(n-1) .
```

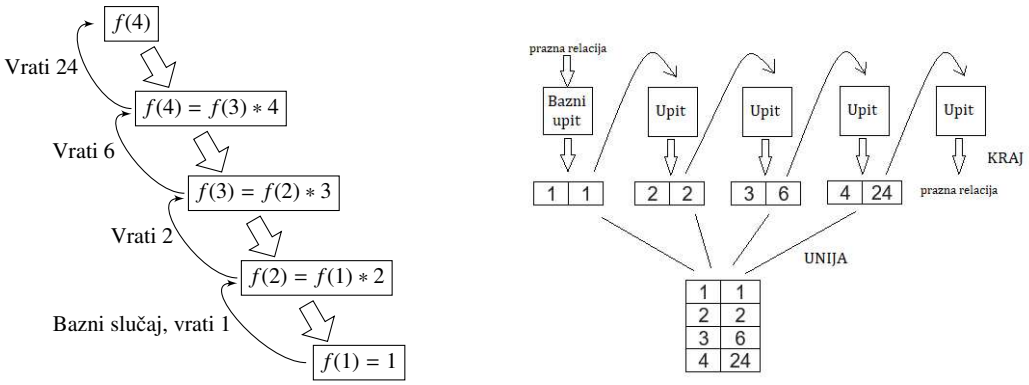
Skica izvođenja te funkcije za $n = 4$ dana je na slici 2.1a. Pripadni rekurzivni upit izgleda ovako:

```
WITH RECURSIVE Fact(n, rez) AS
    (SELECT 1, 1 UNION
     SELECT n+1, (n+1)*rez FROM Fact WHERE n<4)
SELECT n, rez FROM Fact ,
```

dok je skica izvršavanja dana slikom 2.1b. Uočimo kako se u oba slučaja najprije izračuna bazni upit ili slučaj te se u svakoj sljedećoj iteraciji koriste rezultati iz prošle, no da je redoslijed poziva drugačiji.

Pojam koji ćemo često spominjati u nastavku rada je tranzitivno zatvorenje grafa. **Tranzitivno zatvorenje grafa** je skup svih parova vrhova oblika (start, kraj) gdje

[3] Konvergencijom prema praznoj rezultantnoj relaciji bavit ćemo se u točki 3.6.



(a) Izvršavanje rekurzivne funkcije (b) Izvršavanje rekurzivnog upita

Slika 2.1: Skice izvršavanja

je moguće slijediti bridove u grafu od vrha `start` do vrha `kraj`. Pojam tranzitivnog zatvorenja važan je u ispitivanju granica izražajnosti relacijske algebre i jezika SQL. Nije moguće zapisati upit u relacijskoj algebri niti jeziku SQL kojim bismo odredili tranzitivno zatvorenje grafa; to ćemo detaljno argumentirati u poglavlju 3. Uvođenjem rekurzije u SQL3 rješava se nemogućnost postavljanja upita vezanih i uz problem dohvatljivosti (eng. *reachability)*. Taj se problem odnosi na provjeru je li iz zadanog stanja ili vrha moguće transformacijama ili putevima doći do nekog drugog zadanog stanja ili vrha.

Međutim, postoje ograničenja na korištenje rekurzije, a i konkretni razlozi za to. Nepoštivanje svojstava koje rekurzivni upit mora zadovoljavati može dovesti do nemogućnosti izvršavanja ili pogrešnog rezultata, u točki 3.6 reći ćemo nešto više o tome.

Primjer 2.3.1. *Pokažimo kako u SQL-u možemo zapisati rekurzivne upite iz praktičnih primjera koje smo naveli. Obratit ćemo pažnju na sličnost između prva dva primjera, a to je njihova povezanost s problemom traženja putova u usmjerenom grafu.*

Prvi primjer preuzet je iz [12, točka 10.2.1] i vezan je uz letove. Potrebno je pronaći sve gradove koji su direktno povezani jednim letom ili (indirektno) više njih. Relacijska shema baze koja se koristi izgleda ovako: `Letovi (Od, Do)`*, a relacija je prikazana u tablici 2.13. Upit koji tražimo glasi ovako:*

```
WITH RECURSIVE Letovi_presjedanje(Od, Do) AS
     (SELECT Od, Do FROM Letovi UNION
      SELECT R1.Od, R2.Do
      FROM Letovi_presjedanje R1, Letovi R2
      WHERE R1.Do = R2.Od)
 SELECT * FROM Letovi_presjedanje .
```
(□)

Analizirajući upit primjećujemo kako se popis gradova puni iterativno. Najprije se pronađu oni gradovi koji su povezani direktnim letom; na taj slučaj gledamo kao bazni slučaj rekurzije. Parovi gradova dodani u ovom koraku iteracije su u tablici 2.14 obojeni plavom bojom. Nadalje, prethodno pronađeni parovi koriste se pri daljnjem traženju letova, odnosno tražimo gradove koji su povezani s najmanje dva leta između sebe (obojeni crvenom bojom u tablici), pa najmanje tri i tako dalje. Pretraga staje kada u nekoj od iteracija ne nađemo niti jedan par gradova. Primijetimo da smo zapravo tražili tranzitivno zatvorenje grafa gdje su gradovi vrhovi grafa, a direktni letovi bridovi grafa.

Od	Do
San Francisco	Denver
San Francisco	Dallas
Denver	Chicago
Denver	Dallas
Dallas	Chicago
Dallas	New York
Chicago	New York

Tablica 2.13: Tablica `Letovi`

Od	Do
San Francisco	Denver
San Francisco	Dallas
Denver	Chicago
Denver	Dallas
Dallas	Chicago
Dallas	New York
Chicago	New York
San Francisco	Chicago
San Francisco	New York
Denver	New York

Tablica 2.14: Rezultat rekurzivnog upita

Roditelj	Dijete
Alice	Carol
Bob	Carol
Carol	Dave
Carol	George
Dave	Mary
Eve	Mary
Mary	Frank

Tablica 2.15: Tablica `RoditeljOd`

Roditelj
Mary
Dave
Eve
Carol
Alice
Bob

Tablica 2.16: Rezultat rekurzivnog upita

Sljedeći primjer (preuzet iz [20]) često se koristi kao motivacija za uvođenje rekurzije, a opet se svodi na traženje puta u grafu. Bavimo se problematikom traženja predaka zadane osobe koristeći obiteljsko stablo. Osobe reprezentiramo kao vrhove grafa, a bridove usmjerimo od roditelja prema djetetu. Shema relacije je `RoditeljOd(Roditelj, Dijete)`*, a u tablici 2.15 je za svakog roditelja ispisano njegovo dijete. Ako želimo pronaći roditelje, baku i djeda, prabaku i pradjeda određene osobe, napisat ćemo upit fiksne dubine, dakle*

bez upotrebe rekurzije. No, ne možemo izbjeći neki oblik iteracije ili rekurzije ako želimo pronaći sve pretke neke osobe. Iako možemo napisati upit za traženje roditelja, bez rekurzije ne možemo napisati upit u kojem bismo iterativno tražili roditelje zadanih ili prethodno nađenih osoba, gdje bi pretraživanje stalo kada u nekoj iteraciji ne bismo pronašli niti jednog roditelja. Razlog tome je nepostojanje elementa jezika koji bi omogućio iterativno izvršavanje uz provjeru navedenog uvjeta zaustavljanja.

Sljedećim rekurzivnim upitom ćemo dobiti sve pretke osobe imena Frank:

```
WITH RECURSIVE Predak AS
    (SELECT Roditelj AS predak_roditelj
     FROM RoditeljOd
     WHERE Dijete = 'Frank' UNION
     SELECT Roditelj FROM Predak, RoditeljOd
     WHERE Predak.predak_roditelj = RoditeljOd.Dijete)
SELECT * FROM Predak .
```

Slično kao kod letova, u baznom slučaju tražimo Frankove roditelje kao direktne pretke, a potom tražimo roditelje tih predaka, i tako redom dalje. Predci pronađeni u različitim iteracijama obojeni su različitom bojom u tablici 2.16. S pretragom stanemo kada u nekoj od iteracija ne nađemo niti jednog roditelja nekog od predaka iz prethodne iteracije.

ID	Iznos_glavnice
0	100000
1	300000
2	500000

Tablica 2.17: Tablica `Glavnice`

Idući primjer pokazuje korisnost rekurzije prilikom složenih izračuna s više koraka. Baza podataka u ovom primjeru sadrži tablicu `Glavnice(ID, Iznos_glavnice)` u koju ćemo spremati moguće iznose glavnice koje ćemo upotrijebiti za složeni kamatni račun. Podsjetimo se pojmova vezanih uz složeni kamatni račun:

- glavnica *je početni iznos koji stavljamo na račun na početku štednje, oznaka* C_0,
- kamata *je naknada koju banka isplaćuje štediši za uloženi novac u banku, oznaka* k,
- godišnja kamatna stopa *je omjer godišnjeg iznosa kamata i glavnice, oznaka* p,
- *nakon* n *godina štediša će na računu imati vrijednost izračunanu po formuli* $C_n = C_0(1 + p)^n$ *te*
- *ukupni iznos kamate je jednak* $k = C_n - C_0$.

Godine	Iznos_glavnice	Na_racunu	Kamate
5.0	0.00	0.00	0.00
6.0	100000.00	119405.23	19405.23
7.0	100000.00	122987.39	22987.39
8.0	100000.00	126677.01	26677.01
9.0	100000.00	130477.32	30477.32
10.0	100000.00	134391.64	34391.64
6.0	300000.00	358215.69	58215.69
7.0	300000.00	368962.16	68962.16
8.0	300000.00	380031.02	80031.02
9.0	300000.00	391431.96	91431.96
10.0	300000.00	403174.91	103174.91
6.0	500000.00	597026.15	97026.15
7.0	500000.00	614936.93	114936.93
8.0	500000.00	633385.04	133385.04
9.0	500000.00	652386.59	152386.59
10.0	500000.00	671958.19	171958.19

Tablica 2.18: Rezultat izračuna složenog kamatnog računa

Pretpostavimo da je kamatna stopa 3% i na raspolaganju imamo više mogućnosti iznosa glavnice (tablica 2.17). Zanima nas koliko ćemo imati novaca na računu i pripadne kamate u ovisnosti o glavnici te vremenu štednje. Iako formula za računanje nije teška, računanje iznosa u svim mogućim kombinacijama bilo bi vremenski zahtjevno, stoga upitom (ne optimalnijim u pogledu broja računskih operacija) demonstriramo korisnost rekurzije u složenim računima. Rezultati su vidljivi u tablici 2.18, a upit izgleda ovako:

```
WITH RECURSIVE Kamatni_racun AS
  (SELECT 5 AS Godine, 0 AS Iznos_glavnice, 0 AS Na_racunu UNION
   SELECT Godine+1, Glavnice.Iznos_glavnice,
          ROUND(Glavnice.Iznos_glavnice * POWER(1.03,Godine+1),2)
          AS Na_racunu
   FROM Glavnice, Kamatni_racun WHERE Godine < 10)
SELECT *, Na_racunu - Iznos_glavnice AS Kamate
FROM Kamatni_racun ORDER BY Na_racunu;
```

U rezultantnoj tablici upita primjećujemo prvi redak koji je vezan uz bazni upit rekurzije, a gdje su sve vrijednosti postavljenje na početne vrijednosti. Njega po želji možemo ukloniti iz tablice dodavanjem uvjeta `WHERE Godina>5` *u zadnji redak rekurzivnog upita neposredno prije klauzule* `ORDER BY`*.*

Poglavlje 3

Izražajna snaga SQL-a

Ovo poglavlje posvećeno je granicama izražajnosti jezika SQL, preciznije njegovog jezgrenog dijela, detaljno opisanog u poglavlju 2. Proširit ćemo relacijsku algebru kako bismo mogli iskazati svojstva vezana uz izražajnost SQL-a. Na kraju poglavlja proučit ćemo ograničenja prilikom zadavanja rekurzivnih upita u SQL3 standardu, koje smo detaljnije proučavali u točki 2.3.

3.1 Relacijska algebra i SQL

Uvjeti dobre tipiziranosti, učinkovitog izračunavanja i generičnosti, spomenuti u uvodu, utječu na izražajnost jezika za postavljanje upita. Iako tu izražajnost možemo promatrati na više načina, nas će u ovome radu zanimati koje je konkretne ideje i koncepte u semantičkom smislu moguće izraziti u jeziku. Primjerice, zanima nas je li moguće zapisati upit kojim ćemo dobiti tranzitivno zatvorenje neke relacije. Prisjetimo se primjera 2.3.1 i problema traženja predaka, gdje smo napomenuli kako takav upit bez rekurzije u SQL-u nije moguće zapisati. Kao granice izražajnosti nekog jezika navest ćemo elemente koje ne možemo dobiti ili izračunati koristeći jezik, a pritom ćemo pokušati dokazati zašto je tome tako te ćemo detaljnije argumentirati tvrdnju iz uvoda o postavljanju granica izražajnosti SQL-a na temelju granica relacijske algebre.

Tvrdnja o nemogućnosti postavljanja upita za računanje tranzitivnog zatvorenja, koja je dokazana za relacijsku algebru [7, dodatak], ponekad se u literaturi koristi kao argument prilikom iskazivanja granica izražajnosti SQL-a. Međutim, to zaključivanje nije korektno; u nastavku ćemo dati jednostavnu i kratku argumentaciju (iz [16]) zašto.

Nakon pregleda relacijske algebre i SQL-a u poglavljima 1 i 2, primjećujemo da SQL nadograđuje relacijsku algebru. Agregatne funkcije uz operatore grupiranja povećavaju izražajnost SQL-a u odnosu na relacijsku algebru. Jednostavan primjer za to je upit kojim uspoređujemo kardinalitet relacija. U SQL-u upit kojim testiramo ima li relacija `R` veći

kardinalitet od relacije S (ako ima, rezultantna vrijednost upita je 1, a inače je prazna relacija) izgleda ovako:

```
SELECT 1 FROM R WHERE
(SELECT COUNT(*) FROM R) > (SELECT COUNT(*) FROM S),
```

dok takav upit nije moguće izraziti u relacijskoj algebri. Razlog tome je nemogućnost *brojanja* u relacijskoj algebri. Odnosno, ne postoji agregatna funkcija poput COUNT, niti možemo iterativno uzimati jedan po jedan element relacija sve dok neka od relacija ne postane prazna. Precizno obrazloženje zašto ne postoji neki drugi način iskazivanja željenog upita bazira se na relacijskom računu i teorijskom razmatranju svojstava logike prvog reda koja je u pozadini, a dano je u [4].

U nastavku rada ćemo proširiti relacijsku algebru elementima SQL-a koje ne možemo izraziti u relacijskoj algebri kako bi njezina izražajna snaga postala bliža onoj jezika SQL, a zatim ćemo se baviti izražajnošću tog proširenja.

3.2 Proširenje relacijske algebre

U ovoj ćemo točki proširiti operatore relacijske algebre agregatnim funkcijama, grupiranjem i aritmetičkim operacijama (prema [16]) tako da njezina izražajna snaga postane ekvivalentna SQL-ovoj, preciznije jezgrenoj verziji opisanoj u poglavlju 2. Naime, aritmetičke operacije poput zbrajanja i oduzimanja nisu podržane u relacijskoj algebri, dok smo vidjeli da se one smiju pojaviti u SQL-u. Kao i agregatne funkcije (osim COUNT), aritmetičke se operacije računaju na numeričkim operandima. Budući da možemo imati numeričke i nenumeričke atribute, potrebno ih je moći razlikovati.

Tipiziranjem relacije postići ćemo razlikovanje vrste atributa unutar relacije. U našem ćemo slučaju imati samo dvije vrste atributa: numerički i nenumerički. Numerički tip označavat ćemo oznakom n, dok ćemo domenu svih numeričkih atributa označavati s Num; ona može biti instancirana s $\mathbb{N}$, $\mathbb{R}$ i slično. Nenumerički tip imat će oznaku b, dok će domena biti označena s Dom. Definiramo **tip relacije** kao listu vrsta atributa, odnosno riječ nad abecedom {b,n}. Često ćemo umjesto *tip relacije* pisati samo *tip*.

Neka je R relacija tipa t, gdje je t jednak $a_1 a_2 \ldots a_m$. Ona tada ima m stupaca, gdje je a_i, za $i \in \{1, \ldots, m\}$, jednak b ako su vrijednosti i-tog stupca nenumeričke ili n ako su numeričke. Oznaka $t.i$ označavat će i-tu poziciju u tipu (odnosno tip i-tog atributa relacije), dok ćemo duljinu tipa označavati s $|t|$. Shemu baze zapisivat ćemo kao skup imena relacija i njezinih pripadnih tipova; pritom relaciju R tipa t zapisujemo kao $R : t$. Svaki upit poistovjetit ćemo s rezultantnom relacijom i definirati **tip upita** kao tip njegove rezultantne relacije.

Primjer 3.2.1. *Iskoristimo bazu podataka iz primjera 1.1.1 kako bismo oprimjenili prethodno definirane pojmove vezane uz tipiziranje relacije.*

Shema baze izgledala bi ovako: `Knjige : bbbbnb`, `Autori : bbb`, `Izdavaci : bb`. *Pogledajmo detaljnije relaciju* `Knjige`. *Njezin tip je* `bbbbnb`, *što znači da ona ima 6 atributa od kojih je peti numerički, a svi ostali atributi su nenumerički. Ako imamo* $t =$ `bbb`, *izraz* $t.2$ *označava tip drugog stupca koji je u ovom slučaju* `b`, *a vrijedi i* $|t| = 3$. *Tip upita 1.1 je* $t =$ `bn` *jer su stupci rezultantne relacije prikazane u tablici 1.5 redom nenumeričkog i numeričkog tipa.*

Slijedi opis proširene relacijske algebre parametrizirane skupom funkcija, predikata i agregata definirana po L. Libkinu iz [16], koristeći prethodno opisano tipiziranje i djelovanje operatora standardne relacijske algebre (kraće RA) iz poglavlja 1. Uz to se u nastavku koristi oznaka m za duljinu tipa t, $m = |t|$. *Strogo rastući* konačni niz k prirodnih brojeva iz skupa $\{1, \ldots, m\}$ označavat ćemo s $i_1, \ldots, i_k$. Također, koristit ćemo oznaku $\cdot$ za operaciju konkatenacije *stringova* koju ćemo koristiti nad tipovima, npr. izraz $\texttt{bn} \cdot \texttt{nbn}$ označavat će tip `bnnbn`. Najčešće ćemo m-torku $(a_1, \ldots, a_m)$ kraće označavati s $\vec{a}$.

U definiciji 3.2.2 operatori i izrazi su razvrstani prema tome pripadaju li izrazima standardne relacijske algebre (oznaka pored definicije je $\bullet$), aritmetičkim operatorima (oznaka $\square$) ili operatorima agregacije i grupiranja (oznaka $\blacksquare$).

Definicija 3.2.2. ***Proširena relacijska algebra s agregatima*** *nad shemom baze SB, u oznaci* $\mathrm{ALG}_{\mathrm{aggr}}(\Omega, \Theta)$, *gdje je* Ω *skup funkcija i predikata s domenom* `Num` *te* Θ *skup agregatnih funkcija, definirana je sljedećim izrazima i operatorima:*

- ***shema relacije****: ako se* $R : t$ *nalazi u shemi SB, onda je R relacija tipa t. Kraće ćemo pisati* $R : t \in SB$,

- ***skupovni operatori***[1]*: ako su R i S relacije tipa t, tada su i rezultantne relacije* $R \cup S$, $R \cap S$ *i* $R \setminus S$ *tipa t, a djelovanje operatora je kao u RA,*

- ***projekcija***[2]*: ako je* $R : t \in SB$, *tada je tip relacije* $\pi_{i_1,\ldots,i_k}(R)$ *jednak* $t.i_1 \cdot \cdots \cdot t.i_k$, *a djelovanje je isto kao u RA,*

- ***selekcija***[3]*: ako je* $R : t \in SB$ *te* i, j *takvi da vrijedi* $t.i = t.j$, *tada je tip relacije* $\sigma_{i=j}(R)$ *jednak t, a djelovanje je jednako kao u RA,*

[1] Primijetimo kako je uvjet kompatibilnosti dviju relacija za skupovne operacije u ovoj definiciji zadovoljen ukoliko dvije relacije imaju isti tip.

[2] U RA su se u projekcijskoj listi zadavala imena stupaca, a sada zadajemo redne brojeve stupaca.

[3] U RA su se u selekcijskom uvjetu zadavala imena stupaca te uvjet nije morao nužno biti jednakost. U ovoj definiciji u selekcijskom uvjetu zadaju se redni brojevi stupaca čije vrijednosti moraju biti jednake.

- ***Kartezijev produkt**: ako su $R : t_1 \in SB$ i $R : t_2 \in SB$, tada je $R \times S$ relacija tipa $t_1 \cdot t_2$, a djelovanje je isto kao u RA,*

- ***permutacija**: ako je $R : t \in SB$ i θ je neka permutacija od $\{1, \ldots, m\}$, tada je djelovanje permutacije dano izrazom $\rho_\theta(R)$ i rezultat je relacija tipa $t.\theta(1) \cdot \cdots \cdot t.\theta(m)$,*

□ ***numerička selekcija**: ako je $P \subseteq \mathtt{Num}^k$ k-mjesni numerički predikat[4] iz Ω i za niz $i_1, \ldots, i_k$ vrijedi $t.i_j = \mathtt{n}$ za svaki $j \in \{1, \ldots, k\}$, tada je tip relacije $\sigma[P]_{i_1,\ldots,i_k}(R)$ jednak t za svaku relaciju R tipa t, dok će se u rezultantnoj relaciji nalaziti m-torka $(a_1, \ldots, a_m) \in R$ ako i samo ako vrijedi $P(a_{i_1}, \ldots, a_{i_k})$,*

□ ***funkcijski poziv**: ako je $f : \mathtt{Num}^k \to \mathtt{Num}$ funkcija iz Ω, za niz $i_1, \ldots, i_k$ vrijedi $t.i_j = \mathtt{n}$, za svaki $j \in \{1, \ldots, k\}$ i $R : t \in SB$, tada je $Apply[f]_{i_1,\ldots,i_k}(R)$ relacija tipa $t \cdot \mathtt{n}$ koja za svaku m-torku $(a_1, \ldots, a_m) \in R$ sadrži $(a_1, \ldots, a_m, f(a_{i_1}, \ldots, a_{i_k}))$ (i ništa više),*

□ ***konstante**: ako je c numerička konstanta, onda je $Apply[c]_\varepsilon$ relacija tipa $\mathtt{n}$ i jednaka je $\{c\}$,*

■ ***agregacija**[5]: za svaki $\mathcal{F}$ agregat iz Θ, relaciju $R : t \in SB$ i $i \in \mathbb{N}$ takav da vrijedi $t.i = \mathtt{n}$, relacija $Aggr[i : \mathcal{F}](R)$ je tipa $t \cdot \mathtt{n}$. Opišimo djelovanje: neka je R relacija s m-torkama $\vec{a}_1, \ldots, \vec{a}_p$, gdje je $p \in \mathbb{N}$ i $\vec{a}_j = (a_j^1, \ldots, a_j^m)$, za svaki $j \in \{1, \ldots, p\}$; tada $Aggr[i : \mathcal{F}](R)$ zamjenjuje svaku m-torku $\vec{a}_j$ iz R s $\left(a_j^1, \ldots, a_j^m, \mathcal{F}(\{\!\!\{ a_1^i, \ldots, a_p^i \}\!\!\})\right)$,*

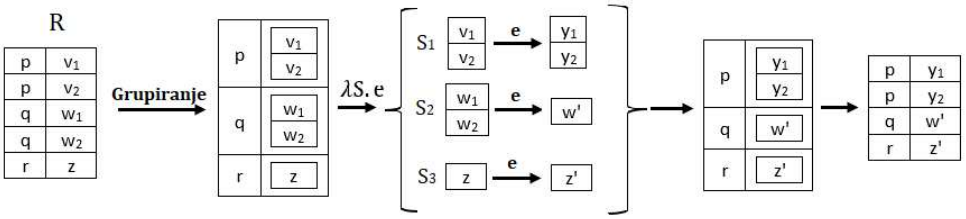
■ ***grupiranje**: neka je e upit tipa t koji se poziva nad relacijom S tipa s te $R : u \cdot s \in SB$ tako da vrijedi $|u| = l$, $l \in \mathbb{N}$; tada je relacija $Group_l[\lambda S.e](R)$ tipa $u \cdot t$. Opisno[6] rečeno, operator grupira m-torke relacije R prema vrijednostima prvih l atributa i izvrednjava upit e na vrijednostima koje su dobivene unutar grupiranja u oznaci S, a to izvrednjavanje označavamo s $\lambda S.e$. Formalni opis djelovanja glasi: neka je R relacija s m-torkama $\vec{a}_1, \ldots, \vec{a}_p$ gdje je $p \in \mathbb{N}$ i $\vec{a}_j = (a_j^1, \ldots, a_j^m)$, za svaki $j \in \{1, \ldots, p\}$; svaku m-torku $\vec{a}_j \in R$ podijelimo na l-torku $\vec{a}_j{}' = (a_j^1, \ldots, a_j^l)$ sastavljenu od prvih l atributa m-torke $\vec{a}_j$ i $(m-l)$-torku $\vec{a}_j{}'' = (a_j^{l+1}, \ldots, a_j^m)$ s preostalim atributima m-torke $\vec{a}_j$; za svaki $\vec{a}_j$ definiramo skup $S_j = \{\vec{a}_r{}'' \mid \vec{a}_r{}' = \vec{a}_j{}', r \in \mathbb{N}\}$, te skup $T_j = \{\vec{b}_j^{\,1}, \ldots, \vec{b}_j^{\,m_j}\}$ čiji su elementi dobiveni izvrednjavanjem upita e nad S_j. Operator $Group_l[\lambda S.e](R)$ vraća skup sastavljen od $(\vec{a}_j{}', \vec{b}_j^{\,i})$ za sve i, j takve da $1 \leqslant j \leqslant p$ i $1 \leqslant i \leqslant m_j$.*

---

[4] Nastavno na definiciju predikata (fusnota 3), numerički predikat je podskup Kartezijevog produkta skupova s numeričkim vrijednostima.

[5] Koristimo matematički zapis agregatnih funkcija prikazan u odjeljku o agregatnim funkcijama.

[6] Za lakše razumijevanje djelovanja grupiranja na slici 3.1 dan je primjer u kojem je $l = 1$.



Slika 3.1: Skica djelovanja operatora grupiranja

Napomena 3.2.3. *U prethodnoj definiciji agregatne funkcije smo definirali samo na numeričkim vrijednostima, no to ne predstavlja ograničenje s obzirom na to da smo u formalnoj definiciji agregatnih funkcija u SQL-u (na kraju točke 2.1) promatrali funkcije s numeričkom domenom. Iako zbog definicije ne možemo koristiti agregatnu funkciju* `COUNT` *za brojanje n-torki u unarnoj nenumeričkoj relaciji, u definiranoj proširenoj relacijskoj algebri postoji način kako dobiti kardinalnost takve relacije [16, točka 3.3].*

Umjesto operatora selekcije iz RA, u $\mathrm{ALG}_{\mathrm{aggr}}$ *smo definirali dva operatora: selekciju i numeričku selekciju. Iako se definicija selekcije čini „stroža" zbog selekcijskog uvjeta (za traženje jednakosti dva stupca), zapravo uz komponiranje sa skupovnim operatorima možemo tražiti proizvoljne kombinacije nejednakosti i jednakosti među stupcima.*

Primjer 3.2.4. *U ovom ćemo primjeru prikazati pozivanje i djelovanje novodefiniranih elemenata u proširenoj relacijskoj algebri prema definiciji 3.2.2, koristeći pritom bazu podataka iz primjera 1.1.1. Grupirat ćemo knjige po autorima i izračunati prosječnu cijenu njihovih knjiga. Odgovarajući upit u SQL-u izgleda ovako:*

```
SELECT ID_autor, AVG(Cijena_kn)
FROM Knjige, Cjenik WHERE Knjige.ISBN = Cjenik.ISBN
GROUP BY ID_autor ,
```

dok ćemo odgovarajući upit u $\mathrm{ALG}_{\mathrm{aggr}}$ *„graditi" postupno. Na početku, želimo dobiti ekvivalent theta-spoja iz relacijske algebre (definicija 1.2.9) koristeći selekciju i Kartezijev produkt, a uvjet spoja će biti jednakost vrijednosti zajedničkog atributa* `ISBN`*. Dodatno, projekcijom ćemo prikazati samo one stupce koji će nam biti potrebni kasnije u upitu. Sve do sad opisano dobit ćemo upitom:*

$$\pi_{3,8}\left(\sigma_{1=7}\left(\texttt{Knjige} \times \texttt{Cjenik}\right)\right), \tag{3.1}$$

a rezultat je prikazan u tablici 3.1.

Definirat ćemo upit koji ćemo koristiti prilikom grupacije. Za to će nam biti potrebna agregatna funkcija $\mathcal{A}$ *definirana u odjeljku o agregatnim funkcijama iz točke 2. Upitom* $Aggr[1 : \mathcal{A}](S)$*, gdje je* S *neka relacija, računamo prosječnu vrijednost u prvom stupcu od* S*.*

Grupirat ćemo rezultantnu relaciju upita 3.1 po prvom stupcu, a potom ćemo izvršiti prethodno opisani upit nad vrijednostima unutar grupiranja. Konačni upit izgleda ovako:

$$Group_1[\lambda S.Aggr[1:\mathcal{A}](S)](\pi_{3,8}(\sigma_{1=7}(\texttt{Knjige} \times \texttt{Cjenik}))), \tag{3.2}$$

a konačni rezultat je dan u tablici 3.2.

ID_autor	Cijena_kn
0001	110
0001	144
0002	199
0002	199
0002	199
0003	50
0004	79
0004	79
0005	129
0005	129

Tablica 3.1: Rezultat upita 3.1

ID_autor	AVG(Cijena_kn)
0001	127
0002	199
0003	50
0004	79
0005	129

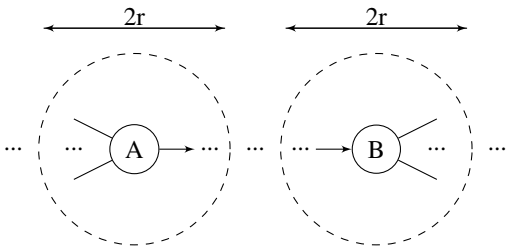
Tablica 3.2: Rezultat upita 3.2

3.3 Lokalnost SQL upita

Granice izražajnosti SQL-a koje smo do sad postavili bile su vezane uz rekurzivne upite, a najviše smo pažnje obratili na tranzitivno zatvorenje relacije. U nastavku ćemo se baviti konkretnijom tvrdnjom o izražajnoj moći SQL-a preuzetoj iz [16]; upiti koji se mogu izraziti u SQL-u su *lokalni*.

Sasvim općenito, pojam **lokalnosti** u logici se često koristi kod dokazivanja izražajnosti jezika [17]. Isto tako, javlja se i u mnogim drugim područjima matematike i računarstva, poput teorije složenosti, formalnih jezika, topologije i slično. Napominjemo da postoje dva pristupa pojmu lokalnosti u literaturi: Hanfova lokalnost i Gaifmanova lokalnost, a za sličnosti i razlike, kao i formalne definicije, upućujemo na [8] i [17]. Mi ćemo u ovom radu koristiti Gaifmanov pristup lokalnosti; Gaifman je prvi promatrao lokalna svojstva upita (prema [11]).

Lokalnost upita u SQL-u vezana je uz to da upit može koristiti i utjecati na samo mali dio polazne relacije. Intuitivno, to znači da odluka o tome pripada li neka n-torka rezultantnoj relaciji upita ovisi o malom broju drugih n-torki, odnosno malom susjedstvu promatrane n-torke neke fiksirane veličine, tzv. fiksiranog radijusa. Lakše je zamisliti primjere upita u kojima moramo pogledati većinu ili sve n-torke relacije kako bismo odredili



Slika 3.2: Skica grafa s vrhovima A i B

je li zadana n-torka u rezultantnoj relaciji, što takav upit čini nelokalnim. Neki primjeri nelokalnih upita su: acikličnost, k-bojanje, traženje iste generacije [16], itd.

Za detaljniji primjer nelokalnog upita ćemo dati problem dohvatljivosti (eng. *reachability*) na grafu. Za vrh B reći ćemo da je *dohvatljiv* iz vrha A ako postoji put u grafu iz vrha A u vrh B. Kažemo da *razlučujemo* n-torke $\vec{x}$ i $\vec{y}$ ako se jedna od njih nalazi u rezultantnoj relaciji zadanog upita, a druga ne. Zadana je relacija koja predstavlja usmjereni graf, npr. sheme G(start, kraj, duljina), gdje je duljina duljina brida između vrha start i vrha kraj. Neka je $r > 0$ fiksirani realni broj koji će predstavljati radijus. Pretpostavimo da u grafu, predstavljenom relacijom G, imamo dva različita vrha A i B. Neka je iz A moguće doći u B i neka je duljina najkraćeg puta iz A u B barem $2r$. Pretpostavimo i da iz B nije moguće doći u A, tj. da A nije dohvatljiv iz B. Kada bi upit e kojim provjeravamo dohvatljivost vrhova u grafu bio lokalan, prema neformalnom opisu lokalnosti, upit bi utjecao i provjeravao samo mali broj vrhova oko zadanog vrha. Preciznije, za vrhove izvan susjedstva radijusa r vrha A ne bi se provjeravalo zadovoljavaju li zadani upit. Analogno vrijedi i za vrh B, prema skici 3.2. To znači da upit ne bi razlučivao n-torku (A,B) od (B,A) jer obje n-torke ne bi bile u rezultantnoj relaciji tog upita. Međutim, isti taj upit e mora razlučivati n-torku (A,B) od (B,A). Razlog tome je pretpostavka o postojanju puta iz vrha A u vrh B, što znači da je B dohvatljiv iz vrha A i zato se u rezultantnoj relaciji tog upita mora nalaziti n-torka (A,B). No, n-torka (B,A) se ne nalazi u rezultantnoj relaciji upita, što znači da bi ovaj upit razlučivao te dvije n-torke i time ne može biti lokalan.

Formalno ćemo definirati lokalnost upita u SQL-u jer će se na tom svojstvu temeljiti iskaz i dokaz teorema o izražajnoj snazi SQL-a. U nastavku koristimo definiranu proširenu relacijsku algebru iz prethodne točke sa svim dozvoljenim aritmetičkim i agregatnim funkcijama (oznaka $\mathrm{ALG}_{\mathrm{aggr}}$(All, All)).

Definicija 3.3.1. *Za shemu baze SB kažemo da je* ***čisto relacijska*** *ako nema niti jednog numeričkog atributa. Analogno se definira čisto relacijska shema relacije i upita.*

Primjer 3.3.2. *Iz primjera 3.2.1 vidimo da su relacije* Autori *i* Izdavači *čisto relacijske. Čisto relacijski upiti su primjerice 2.1 i 2.3.*

Kako bismo lakše razumjeli pojam lokalnosti, u sljedećoj definiciji ćemo promatrati samo upite na grafovima. To znači da relacija predstavlja usmjereni graf, a upitima rješavamo probleme na grafu ili analiziramo svojstva grafa. Izvrednjavanje upita e nad zadanom relacijom R označavamo $e(R)$.

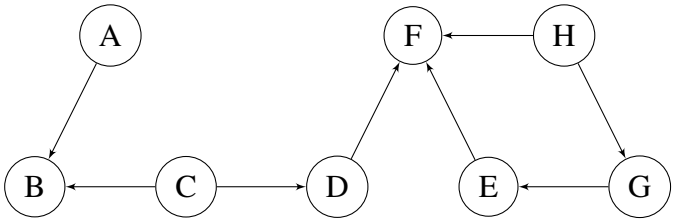
Definicija 3.3.3. *Pretpostavimo da je zadana čisto relacijska shema* R : bb*, gdje* R *sadrži bridove usmjerenog grafa. Za zadani par vrhova* (a, b) *iz* R *i realni broj* $r > 0$*,* ***r-susjedstvo*** *od* a, b*, u oznaci* $N_r^R(a, b)$*, je podskup n-torki iz* R *koje predstavljaju podgraf na vrhovima iz* R *takvim da je udaljenost*[7] *od* a *ili* b *najviše* r*.*

Uz oznake kao prije, za vrhove a, b, c *i* d *iz* R *pišemo* $(a, b) \approx_r^R (c, d)$ *kada su dva r-susjedstva* $N_r^R(a, b)$ *i* $N_r^R(c, d)$ *izomorfna, tj. kada postoji izomorfizam* f *grafova takav da vrijedi* $f(a) = c$ *i* $f(b) = d$*.*

Neka je e *čisto relacijski upit nad* R *tipa* bb *i vrhovi* a, b, c *i* d *iz* R*. Kažemo da je upit* e ***lokalan*** *ako postoji realan broj* $r > 0$ *takav da vrijedi: ako* $(a, b) \approx_r^R (c, d)$*, tada* $(a, b) \in e(R) \Leftrightarrow (c, d) \in e(R)$*.*

Primjer 3.3.4. *Pretpostavimo da relacija R predstavlja usmjereni graf sa slike 3.3:*

$$R = \{(A, B), (C, B), (C, D), (D, F), (E, F), (G, E), (H, F), (H, G)\}.$$



Slika 3.3: Usmjereni graf

Odredimo 1-susjedstvo od vrhova D i H:

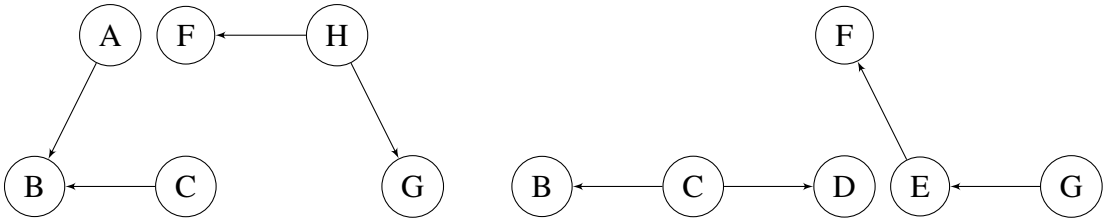
$$N_1^R(D, H) = \{(C, D), (D, F), (H, F), (H, G)\}.$$

Analogno tome, 2-susjedstvo od vrhova A i G je:

$$N_2^R(A, G) = \{(A, B), (C, B), (E, F), (G, E), (H, F), (H, G)\}.$$

Netrivijalni primjer izomorfnih 1-susjedstva je $N_1^R(B, H)$ *i* $N_1^R(C, E)$*, što je vidljivo sa slika 3.4a i 3.4b. Stoga je* $(B, H) \approx_1^R (C, E)$*. Primijetimo,* $N_3^R(C, E) = N_3^R(B, H) = R$ *stoga pišemo* $(C, E) \approx_3^R (B, H)$*. Vrijedi i da je* $N_3^R(C, F) = R$*.*

[7] Udaljenost se određuje na neusmjerenom grafu, to jest promatramo relaciju $R \cup R^{-1}$, gdje je R^{-1} relacija koja za svaku n-torku (x, y) iz R sadrži n-torku (y, x).



(a) $N_1^R(B, H)$ (b) $N_1^R(C, E)$

Slika 3.4: Izomorfna 1-susjedstva

Neka je e upit kojim tražimo sve vrhove ovog grafa koji su dohvatljivi iz vrha C, a rezultantna relacija tog upita sadrži sve parove oblika (C, X)*, gdje je X vrh koji je dohvatljiv iz C. Upit e je primjer nelokalnog upita jer smo pronašli n-torke za koje vrijedi* $(C, F) \approx_3^R (B, H)$*, no ne vrijedi ekvivalencija jer* $(B, H) \notin e(R)$ *i* $(C, F) \in e(R)$*.*

Teorem 3.3.5. *Neka je e čisto relacijski grafovski upit tipa* `bb` *u jeziku* $\mathrm{ALG}_{\mathrm{aggr}}$ $(\texttt{All}, \texttt{All})$ *nad polaznom relacijom* `R:bb`*. Tada je e lokalan upit.*

Ovaj rezultat, jedan od važnijih teorema vezanih uz izražajnu moć SQL-a, može se poopćiti; za to će nam biti potrebna poopćena definicija lokalnosti.

Definicija 3.3.6. *Neka je D čisto relacijska shema baze te neka se baza sastoji od relacija* $R_1, \ldots, R_k$ *mjesnosti* $m_1, \ldots, m_k$ *redom. Definiramo:*

- ***aktivnu domenu relacije*** *R, u oznaci adom*(R)*, kao skup svih elemenata relacije R. Formalno: neka je* $R = \{\vec{a_1}, \ldots, \vec{a_p}\}$ *relacija mjesnosti m, tada je:* $adom(R) = \bigcup_{i \leqslant p} \bigcup_{j \leqslant m} \{a_i^j\}$*;*

- ***aktivnu domenu baze*** *D, u oznaci adom*(D)*, kao skup svih elemenata iz* `Dom` *koji se pojavljuju u relacijama* $R_1, \ldots, R_k$ *u D. Formalno:* $adom(D) = \bigcup_{i \leqslant k} adom(R_i)$*;*

- ***Gaifmanov graf*** *od D kao neusmjereni graf* $\mathcal{G}(D)$ *sa skupom vrhova adom*(D)*, gdje brid između vrhova a i b pripada grafu,* $(a, b) \in \mathcal{G}(D)$*, ako a i b pripadaju istoj n-torci u nekoj relaciji iz D. Formalno: definira se kao neusmjereni graf* $\mathcal{G}(adom(D), E)$*, gdje je* $E = \bigcup_{l \leqslant k} \bigcup_{\vec{a} \in R_l} \{(a^i, a^j) \mid 1 \leqslant i < j \leqslant m_l\}$*;*

- ***udaljenost*** *vrhova* a, b *iz adom*(D)*, u oznaci* $d_D(a, b)$*, kao duljina najkraćeg puta između a i b u Gaifmanovom grafu* $\mathcal{G}(D)$*. Formalno: za* $a, b \in adom(D)$ *definiramo:*

$$d_D(a,b) = \begin{cases} 0, & \text{ako } a = b, \\ \text{duljina najkraćeg takvog puta}, & \text{postoji put između } a \text{ i } b \text{ u } \mathcal{G}(D), \\ \infty, & \text{ne postoji put u } \mathcal{G}(D) \text{ između } a \text{ i } b; \end{cases}$$

- ***r-kuglu*** *oko vrha $a \in adom(D)$, u oznaci $S_r^D(a)$, kao skup svih vrhova iz $adom(D)$ za koje vrijedi da im je udaljenost od vrha a manja ili jednaka od r;*
- ***r-kuglu*** *oko $\vec{a} = (a_1, \ldots, a_k)$ kao $S_r^D(\vec{a}) = \bigcup_{i \leqslant k} S_r^D(a_i)$;*
- ***r-susjedstvo*** *od $\vec{a}$, u oznaci $N_r^D(\vec{a})$, kao novu bazu čija je aktivna domena jednaka $S_r^D(\vec{a})$, a relacije u bazi su restrikcije relacija iz D; formalno, relacije u $N_r^D(\vec{a})$ su:*

$$R_1 \cap \left(S_r^D(\vec{a})\right)^{m_1}, \ldots, R_k \cap \left(S_r^D(\vec{a})\right)^{m_k}.$$

Pišemo $\vec{a} \approx_r^D \vec{b}$ ako postoji izomorfizam $f : N_r^D(\vec{a}) \to N_r^D(\vec{b})$ takav da vrijedi $f(\vec{a}) = \vec{b}$.

*Kažemo da je čisto relacijski upit e **lokalan** ako postoji realni broj $r > 0$ takav da za svaku bazu D vrijedi: ako $\vec{a} \approx_r^D \vec{b}$, onda vrijedi $\vec{a} \in e(D) \Leftrightarrow \vec{b} \in e(D)$. U tom slučaju, najmanji takav r nazivamo **rang lokalnosti** upita e i označavamo s $lr(e)$.*

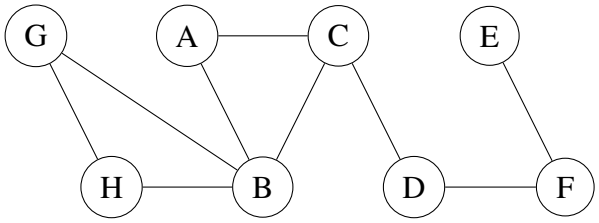
Primjer 3.3.7. *U ovom ćemo primjeru koristiti bazu DB u kojoj se nalaze sljedeće relacije:*

$$R = \{(A, C, B), (B, G, H)\},$$
$$Q = \{(A, B), (D, C), (E, F), (F, D)\}.$$

Aktivne domene relacija R i Q su redom:

$$adom(R) = \{A, B, C, G, H\} \; i$$
$$adom(Q) = \{A, B, C, D, E, F\},$$

dok je $adom(DB) = \{A, B, C, D, E, F, G, H\}$. Gaifmanov graf izgleda kao na slici 3.5.



Slika 3.5: Gaifmanov graf za bazu DB

Koristeći Gaifmanov graf možemo odrediti udaljenosti koje će nam biti potrebne za određivanje r-kugli i r-susjedstva. Primjerice, prema slici 3.5 vidimo da je $d_{DB}(C, H) = 2$, dok je $d_{DB}(A, E) = 4$.

Nadalje, 1-kugle oko vrhova A i H su: $S_1^{DB}(A) = \{A, B, C\}$, $S_1^{DB}(H) = \{B, G, H\}$, dok je 2-kugla oko vrha H jednaka $S_2^{DB}(H) = \{A, B, C, G, H\}$. Unijom prethodno nađenih 1-kugli imamo da je $S_1^{DB}(A, H) = \{A, B, C, G, H\}$.

Odredimo 1*-susjedstvo od (A,B). Primijetimo da vrijedi:*

$$S_1^{DB}(A) = \{A, B, C\} \text{ i } S_1^{DB}(B) = S_1^{DB}(A, B) = \{A, B, C, G, H\}.$$

Aktivna domena susjedstva, tj. nove baze, je jednaka $adom(N_1^{DB}(A, B)) = S_1^{DB}(A, B) = \{A, B, C, G, H\}$*. Relacije u novoj bazi su jednake:*

$$R \cap \left(S_1^{DB}(A, B)\right)^3 = \{(A, C, B), (B, G, H)\},$$
$$Q \cap \left(S_1^{DB}(A, B)\right)^2 = \{(A, B)\}.$$

Uočimo trivijalni izomorfizam između 1-susjedstva $N_1^{DB}(A, B)$ *i* $N_1^{DB}(A, H)$ *jer vrijedi* $S_1^{DB}(A, B) = S_1^{DB}(A, H) = \{A, B, C, G, H\}$ *pa vrijedi* $(A, B) \approx_1^{DB} (A, H)$*.*

Tranzitivno zatvorenje relacije Q je najmanja relacija koja sadrži Q i tranzitivna je. Upitom e želimo dobiti tranzitivno zatvorenje relacije Q. Taj je upit primjer nelokalnog upita zbog toga što vrijedi $(A, B) \approx_1^{DB} (A, H)$*, no nije istinita ekvivalencija iz definicije 3.3.6 jer* $(A, B) \in e(DB)$*, ali* $(A, H) \notin e(DB)$*. Isti zaključak vrijedit će za bilo koji* $r > 0$*, to lako vidimo kada uzmemo dovoljno veliki r da aktivna domena u r-susjedstvu bude jednaka* $adom(DB)$ *i relacije budu jednake R i Q i tada ekvivalencija ponovo neće vrijediti za npr.* (A, B) *i* (A, H)*.*

Teorem 3.3.8. *Neka je e čisto relacijski upit u* $\mathrm{ALG}_{\mathrm{aggr}}(\mathtt{All}, \mathtt{All})$ *nad čisto relacijskom shemom. Tada je e lokalan.*

Prethodno navedeni rezultati pokazuju nemogućnost postavljanja upita u jezgrenoj verziji SQL-a kojim bismo računali tranzitivno zatvorenje grafa. U idućoj točki ćemo dati skicu dokaza prethodnog teorema (detalji se nalaze u [16]).

3.4 Logika i SQL

Autor C. J. Date je u [10] napisao često citiranu rečenicu:

> „Logika i baze podataka neraskidivo su isprepletene".

Ovu tvrdnju možemo potkrijepiti brojnim činjenicama. Neke od njih smo već spominjali u radu, poput veze SQL-a i relacijskog računa (koji je baziran na logici prvog reda), kao i svojstvo lokalnosti koje promatramo u upitima, a dolazi iz logike. Isto tako, teorija konačnih modela [18] je grana logike koja je važna za proučavanje jezika za postavljanje upita u bazama podataka jer služi kao alat za dokazivanje izražajnosti i složenosti.

U [16] Libkin prezentira pregled najznačajnijih dotadašnjih pokušaja dokazivanja granica izražajnosti SQL-a. Također, tvrdi da su komplikacije vezane uz uvođenje kompleksnih jezika nepotrebne i iznosi jednostavan dokaz o granicama izražajnosti SQL-a koristeći

tehnike i rezultate iz logike, konkretno, koristeći jezike bazirane na logici. Razlog tome je taj što se korištenje logike, logičkih struktura i tehnika preuzetih iz logike pokazalo veoma korisno i važno prilikom proučavanja jezika za postavljanje upita. Sukladno tome, autor definira dvije logike čiji će značaj biti istaknut u svakom od tri koraka dokazivanja teorema 3.3.8.

Prvi korak: logika $\mathcal{L}_{\text{aggr}}$

U prvom koraku dokaza potrebno je uvesti novu logiku za koju treba pokazati da se svaki upit iz ALG_{aggr} može prevesti u formulu novouvedene logike.

Definiramo logiku $\mathcal{L}_{\text{aggr}}$ uz pretpostavku da je čitatelj upoznat s pojmovima logike prvog reda [18, 23]. Logika $\mathcal{L}_{\text{aggr}}$ je nadogradnja logike prvog reda u pogledu struktura u kojima može biti zadano više nosača. Takva se logika općenito naziva **višesortna logika** (eng. *many-sorted logic*), a za nosače se češće koristi naziv **sorte** ili **vrste** [25, Poglavlje 1]. Mi ćemo zadati signaturu i definirati strukturu za logiku $\mathcal{L}_{\text{aggr}}$ prema [16, 14].

Definicija 3.4.1. *Alfabet logike $\mathcal{L}_{\text{aggr}}$ definiramo zadavanjem skupa nelogičkih simbola, konkretno signature $\sigma = \{R_1, \ldots, R_k\}$, gdje su $R_1, \ldots, R_k$ simboli za relacije čije su mjesnosti redom $m_1, \ldots, m_k$ i tipovi redom $t_1, \ldots, t_k$.*

Bazu podataka možemo promatrati kao višesortnu strukturu gdje su nosači domene atributa relacije. U nastavku ćemo koristiti dvosortnu strukturu vezanu uz relacije, obzirom na to da smo domene atributa podijelili na `Dom` i `Num`, kao redom prvu i drugu sortu. Zatim, $\mathcal{L}_{\text{aggr}}$ ćemo parametrizirati skupom predikata i funkcija na `Num` (oznaka Ω) i skupom agregata (oznaka Θ), u oznaci $\mathcal{L}_{\text{aggr}}(\Omega, \Theta)$.

Definicija 3.4.2. *Definiramo **SB-strukturu** D kao $(k+1)$-torku oblika $(\{A, \texttt{Num}\}, R_1^D, \ldots, R_k^D)$, gdje je A konačan podskup od `Dom` i za svaki $i \in \{1, \ldots, k\}$, relacija[8] R_i^D mjesnosti m_i tipa t_i je konačan podskup od $\Pi_{j=1}^{|t_i|} dom_j(D)$, gdje je $dom_j(D) = A$ za $t_i.j = \texttt{b}$ i $dom_j(D) = \texttt{Num}$ za $t_i.j = \texttt{n}$.*

Definiramo:

- *varijabla prve vrste je term prve vrste, a varijabla druge vrste je term druge vrste;*
- *ako su τ, τ' termi iste vrste, onda je $\tau = \tau'$ formula;*
- *ako je $R : t \in SB$ i $\vec{u}$ n-torka terma tipa t, onda je $R(\vec{u})$ formula;*
- *formule su zatvorene s obzirom na $\wedge, \vee, \neg$ i kvantifikatore;*

[8] Umjesto R_i^D kraće ćemo pisati R_i kada je jasno da se ne radi o relacijskom simbolu već o relaciji iz strukture.

- *ako je x varijabla prve vrste, onda $\exists x$ znači $\exists x \in A$, dok ako je y varijabla druge vrste, onda $\exists k$ znači $\exists k \in \texttt{Num}$;*
- *ako je P n-mjesni predikat iz Ω i $\tau_1, \ldots, \tau_n$ termi druge vrste, onda je $P(\tau_1, \ldots, \tau_n)$ formula;*
- *ako je f n-mjesna funkcija iz Ω i $\tau_1, \ldots, \tau_n$ termi druge vrste, onda je $f(\tau_1, \ldots, \tau_n)$ term druge vrste;*
- *ako je $\mathcal{F}$ agregat iz Θ, $\varphi(\vec{x}, \vec{y})$ formula i $\tau(\vec{x}, \vec{y})$ term druge vrste, tada je $\tau'(\vec{x}) = Aggr_{\mathcal{F}}\, \vec{y}.(\varphi(\vec{x}, \vec{y}), \tau(\vec{x}, \vec{y}))$ term druge vrste sa slobodnim varijablama $\vec{x}$.*

Naposljetku, navodimo rezultat [16] o prevođenju upita iz RA u formulu agregatne logike koji predstavlja završni dio prvog koraka dokaza lokalnosti upita u SQL-u. Njegov je značaj vezan uz prelazak iz algebre u logiku i mogućnost korištenja tehnika i rezultata iz logike. Prevođenje u drugom smjeru, odnosno prevođenje formule agregatne logike u upit relacijske algebre ne vrijedi u potpunosti, već vrijedi parcijalni rezultat [14]. Dokaz teorema je tehničke prirode stoga ga izostavljamo.

Teorem 3.4.3. *Neka je $e : t$ upit iz $\mathrm{ALG}_{\mathrm{aggr}}(\Omega, \Theta)$. Postoji formula $\varphi_e(\vec{x})$ iz $\mathcal{L}_{\mathrm{aggr}}(\Omega, \Theta)$, gdje je $\vec{x}$ tipa t, takva da za bilo koju SB-strukturu D vrijedi:*

$$e(D) = \{\vec{a} \mid D \models \varphi_e(\vec{a})\}.$$

Drugi korak: logika $\mathcal{L}_{\mathrm{C}}$

U drugom koraku dokaza se s jedne strane pojednostavi novouvedena logika uvođenjem logike s brojanjem $\mathcal{L}_{\mathrm{C}}$ koja neće imati terme vezane uz agregaciju. No, to pojednostavljenje s druge strane dovest će do kompliciranije logike od one prethodno definirane, zbog mogućih pojava beskonačnih formula s veznicima $\vee$ i $\wedge$.

Ovu logiku nećemo formalno definirati, već ćemo samo navesti značajnije dijelove definicije. Logika $\mathcal{L}_{\mathrm{C}}$ se promatra kao $\mathcal{L}_{\mathrm{aggr}}(\emptyset, \emptyset)$ [14]. Ona će imati jednaku strukturu kao i logika $\mathcal{L}_{\mathrm{aggr}}$, s dodatnim ograničenjima kod definicija terma i formula [16].

Termi u ovoj logici su varijable bilo koje vrste i konstante iz `Num` koje su termi druge vrste. Atomarne formule su oblika $R(\vec{x})$, gdje je R relacijski simbol, a $\vec{x}$ n-torka terma, i oblika $x = y$, gdje su x, y termi iste vrste. Formule su, kao i u $\mathcal{L}_{\mathrm{aggr}}$, zatvorene s obzirom na $\wedge, \vee, \neg$ i kvantifikatore, no dodatno su zatvorene s obzirom na beskonačne konjunkcije i disjunkcije; neka su φ_i formule za $i \in I$, tada su i $\bigwedge_{i \in I} \varphi_i$ i $\bigvee_{i \in I} \varphi_i$ formule. Važnije proširenje je i uvođenje tzv. brojajućih kvantifikatora (eng. *counting quantifiers*), kako bismo iskazali da „postoji najmanje k elemenata koji zadovoljavaju neko svojstvo". Formalnije, neka je v neka valuacija, D SB-struktura, A konačan podskup od `Dom` i neka je $\varphi(x, \vec{y})$ formula; tada

je $\exists^{i}x\ \varphi(x,\vec{y})$ formula čije su slobodne varijable $\vec{y}$. Vrijedi $D \models_v \exists^{i}x\ \varphi(x,\vec{y})$ ako postoji i različitih elemenata $b_1,\ldots,b_i$ iz A takvih da je $D \models_v \varphi(b_j,\vec{y})$ za $1 \leqslant j \leqslant i$.

Nadalje, potrebno je pokazati da za svaku formulu iz $\mathcal{L}_{\text{aggr}}$ postoji ekvivalentna formula iz $\mathcal{L}_{\text{C}}$, što je dio sljedećeg koraka dokaza.

Treći korak: rang formule i lokalnost u $\mathcal{L}_{\text{C}}$

U zadnjem će koraku dokaza u fokusu biti pojam ranga formule. Njegova je važnost vidljiva u rezultatu o pretvorbi formula iz $\mathcal{L}_{\text{aggr}}$ u $\mathcal{L}_{\text{C}}$, što ćemo iskazati nakon definicije ranga formula. Također, rang će biti ključan dio dokaza o lokalnosti formula iz $\mathcal{L}_{\text{C}}$, čime ćemo završiti dokaz teorema 3.3.8.

Rang formule definira se posebno za obje logike, a mi ćemo u ovom radu formalno iskazati definiciju ranga formule za $\mathcal{L}_{\text{aggr}}$, analogno se definira i za $\mathcal{L}_{\text{C}}$ (pogledati [16]). Rang formule brojat će ugniježđene kvantifikatore po varijablama prve vrste u formuli, što će uočiti u pretposljednjoj stavci sljedeće definicije.

Definicija 3.4.4. *Definiramo **rang**, u oznaci $rk(\cdot)$, ovako:*

- *rang varijable i konstantnog terma je 0;*
- *rang atomarne formule je maksimalni rang terma u njoj;*
- $rk(\varphi_1 * \varphi_2) = max\{rk(\varphi_1), rk(\varphi_2)\}$, *za* $* \in \{\vee, \wedge\}$;
- $rk(\neg\varphi) = rk(\varphi)$;
- $rk(f(\tau_1,\ldots,\tau_n)) = max_{1\leqslant i\leqslant n}\, rk(\tau_i)$;
- $rk(\exists x\varphi) = rk(\varphi) + 1$ *ako je x prve vrste,* $rk(\exists x\varphi) = rk(\varphi)$ *ako je x druge vrste;*
- $rk(Aggr_{\mathcal{F}}\vec{y}.(\varphi,\tau)) = max\{rk(\varphi), rk(\tau)\} + m$, *gdje je m broj varijabli prve vrste u* $\vec{y}$.

Sljedeći dio dokaza ponovo (prisjetimo se 3.4.3) je vezan uz prevođenje iz jednog jezika u drugi, u ovom slučaju iz jedne logike u drugu, koristeći prethodno definirani pojam ranga. Rezultat ćemo samo iskazati, a dokaz se može pronaći u [16, prop. 1].

Neka su zadane dvije logike $\mathcal{L}_1$ i $\mathcal{L}_2$. Ako za svaku formulu u $\mathcal{L}_1$ postoji ekvivalentna formula iz $\mathcal{L}_2$ manjeg ili jednakog ranga pišemo $\mathcal{L}_1 \preccurlyeq_{rk} \mathcal{L}_2$. Ako vrijedi i $\mathcal{L}_1 \preccurlyeq_{rk} \mathcal{L}_2$ i $\mathcal{L}_2 \preccurlyeq_{rk} \mathcal{L}_1$ to zapisujemo kao $\mathcal{L}_1 \approx_{rk} \mathcal{L}_2$.

Propozicija 3.4.5. *Za svaku formulu $\varphi(\vec{x})$ iz $\mathcal{L}_{\text{aggr}}$ postoji ekvivalentna formula $\varphi'(\vec{x})$ iz $\mathcal{L}_C$ manjeg ili jednakog ranga, tj. $rk(\varphi') \leqslant rk(\varphi)$.*

Budući da formule iz $\mathcal{L}_{\text{aggr}}$ imaju konačan rang (ne postoje beskonačne formule s kvantifikatorima) koristeći prethodnu propoziciju zaključujemo i da formule iz $\mathcal{L}_C$ također imaju konačan rang. Kako bi se završio dokaz teorema 3.3.8 potrebno je još pokazati lokalnost formula iz $\mathcal{L}_C$. Za to se koristi tzv. permutacijska lema iz [16] kao pomoćna tvrdnja.

Pokazuje se da svaka formula φ iz $\mathcal{L}_{\text{C}}$ konačnog ranga je lokalna, gdje je minimalni $r \geqslant 0$ iz definicije 3.3.6 ovisi od rangu formule φ, preciznije vrijedi: $lr(\varphi) \leqslant \frac{1}{2} \cdot (3^{rk(\varphi)} - 1)$, gdje je $lr(\varphi)$ definiran u 3.3.6.

Naposljetku, neka je e čisto relacijski izraz u $\text{ALG}_{\text{aggr}}(\texttt{All}, \texttt{All})$. Po teoremu 3.4.3 on je izraziv u logici $\mathcal{L}_{\text{aggr}}(\texttt{All}, \texttt{All})$, a po propoziciji 3.4.5 iskaziv je u logici $\mathcal{L}_{\text{C}}$. Također, dodatno vrijedi da je pripadajuća formula iz $\mathcal{L}_{\text{C}}$ konačnog ranga, što povlači da je lokalna. Time je završen dokaz teorema 3.3.8.

3.5 Zaključak

U ovoj ćemo točki na jednom mjestu sažeti sve tvrdnje i rezultate do sad iskazane o izražajnoj snazi jezgrene verzije SQL-a. Općenito, prilikom proučavanja izražajnosti nekog jezika u fokusu su dva glavna pitanja.

Prvo je pitanje vezano uz to što možemo i, još bitnije, što ne možemo iskazati u jeziku. Proučavajući jezik SQL stekli smo uvid u to koje upite možemo postavljati, a posebnu smo pažnju posvetili rekurzivnim upitima, poput tranzitivnog zatvorenja relacije i problema dohvatljivosti, za koje smo rekli da nisu iskazivi u jezgrenoj verziji SQL-a.

Sljedeće je pitanje vezano uz metode i tehnike kojima možemo dokazati da navedeni elementi nisu iskazivi u jeziku. Istaknimo metodu uspoređivanja izražajnosti dvaju jezika. Naime, relacijsku algebru smo proširili kako bi njena izražajna moć postala bliža onoj jezika SQL. Uvedene su agregatne funkcije, grupiranje i aritmetičke operacije, no potrebno je bilo razlikovati vrste atributa u relacijama, što smo postigli tipiziranjem relacija i upita.

Najvažnija tehnika koju smo koristili u radu je preuzeta iz logike, a odnosi se na svojstvo lokalnosti upita ili formula. Granice izražajnosti precizno smo postavili rezultatom (teorem 3.3.8) koji kaže da su svi upiti koje je moguće izraziti u SQL-u lokalni. Svojstvo lokalnosti definirali smo u općenitosti, kao i pokazali u konkretnom primjeru na grafovskim relacijama; također, vidjeli smo i pripadni teorem o lokalnosti upita na takvim relacijama (teorem 3.3.5).

U nastavku rada smo dali skicu dokaza teorema 3.3.8 koji se provodi u 3 koraka. U prva dva koraka smo uveli nove logike — $\mathcal{L}_{\text{aggr}}$ i $\mathcal{L}_{\text{C}}$, koje zapravo promatramo kao jezike za postavljanje upita. Potrebno je bilo i pokazati da je moguće „prevesti" izraze iz jednog jezika u drugi, pa smo zato iskazali teorem 3.4.3 (iz ALG_{aggr} u $\mathcal{L}_{\text{aggr}}$) i propoziciju 3.4.5 (iz $\mathcal{L}_{\text{aggr}}$ u $\mathcal{L}_{\text{C}}$). Za spomenutu propoziciju, kao i završni dio dokaza, koristi se pojam i svojstva ranga formule.

Zanimljivo je napomenuti da se malom promjenom proširenja relacijske algebre, dopuštajući neke operatore nad numeričkim vrijednostima i promjenom definicije selekcije, iskazivanje izražajnosti veoma komplicira, što rezultira time da svojstvo lokalnosti više nije dovoljno za dokaz neizražajnosti rekurzivnih upita [16, točka 8.]. Ističemo da navedene tehnike i metode nisu jedini alati koji se koriste prilikom proučavanja izražajnosti. Naime, u literaturi [19] se spominju logički alati poput Ehrenfeucht-Fraïsséovih igara, 0-1 zakona i slično.

3.6 Ograničenja rekurzije

U točki 2.3 rečeno je kako uvođenjem rekurzije i dalje postoje granice izražajnosti SQL-a. Razlog tome su ograničenja koja su postavljena na rekurziju u standardu SQL3, a u ovoj ćemo se točki osvrnuti upravo na ta ograničenja.

Prvo takvo ograničenje [12] koje ćemo razmotriti je da se od rekurzije zahtijeva da je **linearna**, odnosno da se unutar definicije rekurzivne funkcije rekurzivna relacija ne smije pojaviti više od jednom. Konkretno, u općem obliku rekurzivnog upita iz točke 2.3 relacija *R* se u <*bazni upit*> i <*rekurzivni upit*> smije pojaviti najviše jednom.

Prisjetimo se rekurzije iz primjera 2.3.1 vezane uz letove i primijetimo kako je napisani rekurzivni upit linearan jer se ime `Letovi_presjedanje` pojavljuje u baznom i rekurzivnom dijelu upita najviše jednom. Međutim, kada bismo u retku označenom s □ zamijenili `Letovi` s `Letovi_presjedanje`, upit više ne bi bio linearan zbog dvostrukog pojavljivanja rekurzivne relacije unutar klauzule `FROM`. Takav upit ne bi bio legalan u SQL3 standardu, iako bismo izvršavajući upit po definiciji dobili jednaki rezultat kao ranije.

Nelinearne rekurzije se često na jednostavan način mogu pretvoriti u linearne. Pokazat ćemo taj način na prethodno promatranom primjeru s letovima. U nelinearnoj verziji upita gdje se `Letovi_presjedanje` pojavljuje dva puta u retku □, jedno pojavljivanje rekurzivne relacije `Letovi_presjedanje` zamijenimo s relacijom `Letovi` i time upit postaje linearan. Navedeni postupak ponekad nije dovoljan da zadovoljimo uvjet linearnosti; tada upit nije iskaziv u SQL3 standardu.

Još jedno svojstvo rekurzije koje je prema standardu nužno osigurati pri njenom zadavanju je monotonost. Neka je Q upit nad relacijom R. Kažemo da je Q **monoton** s obzirom na R ako dodavanje *n*-torki u relaciju R nikad ne izaziva brisanje *n*-torki iz rezultata upita Q. Dakle, dozvoljeno je da se u rezultantnoj relaciji upita Q dodaju *n*-torke ili da ta relacija ostane nepromijenjena.

Primjere monotonih i nemonotonih upita potražit ćemo među već napisanim upitima u radu. Primjer monotonog upita je upit 2.3 jer dodavanjem novih izdavača *n*-torke iz tablice 2.3 ostaju u rezultantnoj tablici. Međutim, pogledajmo upit 2.6. Kada bismo u tablicu 1.1 (`Knjige`) dodali novu knjigu obrazovnog žanra koja je izdana 2019. godine,

upit 2.3 bi vratio praznu tablicu. Taj upit stoga nije monoton s obzirom na relaciju `Knjige` jer bi se n-torka iz tablice 2.6 izbrisala, odnosno ne bi se našla u rezultantnoj tablici.

Kao još jedan od primjera nemonotonih upita možemo navesti upit iz primjera 2.1.5 u kojem računamo minimalnu, maksimalnu i srednju cijenu knjiga; dodavanjem novih knjiga, vrijednosti iz tablice 2.8 bi se zamijenile novim vrijednostima.

Standard SQL3 ne dopušta kršenje svojstva monotonosti u rekurzivnim upitima jer to dovodi do nemogućnosti izvršavanja upita; razlog može biti nejedinstvenost minimalne fiksne točke ili divergencija metode fiksne točke. Prisjetimo se matematičkog pojma fiksne točke, metode fiksne točke i definirajmo pojam fiksne točke rekurzivne relacije.

Neka je $f : \tau \rightarrow \tau$ funkcija, gdje je τ proizvoljan tip. Fiksna točka funkcije f je vrijednost $x \in \tau$ za koju vrijedi da je $f(x) = x$. **Metoda fiksne točke** (poznata i pod nazivom *metoda jednostavne iteracije*) numerička je metoda rješavanja problema oblika $f(x) = x$, a provodi se tako da zadamo početnu vrijednost (aproksimaciju) x_0 i izračunamo $f(x_0)$. Ako vrijedi $f(x_0) = x_0$ postupak staje i rješenje je x_0, a inače vrijednost $f(x_0)$ uzimamo kao početnu vrijednost za sljedeći korak metode, odnosno računamo $f(x_1)$ za $x_1 = f(x_0)$ te provjeravamo vrijedi li jednakost $x_1 = f(x_1)$, i tako dalje. Metoda ne mora nužno konvergirati; tada se mogu zadati neki kriteriji zaustavljanja.

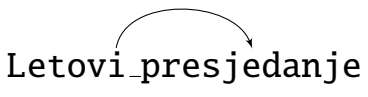
Sada je potrebno pojmove fiksne točke i metode fiksne točke definirati za upite. Vrijedi da svaki upit Q možemo promatrati kao funkciju koja ulaznu tablicu (navedenu nakon ključne riječi `FROM`) preslika u rezultantnu tablicu. **Fiksna točka upita** definira se kao tablica T za koju vrijedi $Q(T) = T$, odnosno upit nad tablicom T kao rezultat daje ponovo tablicu T. Metoda fiksne točke kod upita slična je kao kod funkcija, osim što se ovdje kao početna vrijednost tablice T uzima prazna tablica, zatim se izračuna upit Q nad T i ako rezultat nije T ponavlja se postupak. Odnosno, računa se upit Q, ali nad rezultantnom tablicom upita Q nad T. Kretanje od prazne tablice omogućava nam pronalaženje jedinstvene minimalne fiksne točke, pod pretpostavkom da je upit Q monoton. Naime, zahtjev da se n-torke rezultantne relacije ne brišu prilikom dodavanja novih n-torki u početnu relaciju osigurat će nam minimalnost fiksne točke zbog toga što nakon pronalaska fiksne točke, sljedećim iteracijama ona se može samo puniti novim n-torkama. Izvršavanje upita nad praznom tablicom u početnoj iteraciji osigurava jedinstvenost minimalne fiksne točke. Razlog tome je postupno punjenje rezultantne relacije n-torkama dok ne dođemo do fiksne točke kao i determinizam upita, odnosno činjenica da upit nad nekom tablicom uvijek daje isti rezultat kod svakog izvršavanja.

Postoji još jedan zahtjev na rekurzije, vezan za međusobno rekurzivne relacije. Standardom je moguće dozvoliti definiranje međusobno rekurzivnih relacija u istom rekurzivnom upitu oblika:

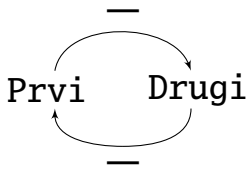
```
WITH
    RECURSIVE R1 AS <definicija od R1>,
    RECURSIVE R2 AS <definicija od R2>,
    ...
    RECURSIVE Rn AS <definicija od Rn>
<upit s R1, R2, ..., Rn>
```

Provjera je li moguće dozvoliti takvu međusobnu ovisnost obavlja se stvarajući **graf ovisnosti** u kojem je svaki čvor jedna od relacija koja se nalazi u `WITH` klauzuli. Vrhovi R i S su povezani usmjerenim bridom iz R u S ako je relacija R definirana preko[9] relacije S. Oznakom '-' označava se brid iz R u S ako upit koji definira relaciju R nije monoton s obzirom na S. Upit u SQL3 standardu ne smije imati ciklus u grafu ovisnosti u kojem se nalazi barem jedan brid označen s '-' jer se takav upit smatra ilegalnim zbog mogućih problema prilikom njegovog izvršavanja.

Vratimo se na primjer 2.3.1 i pogledajmo rekurzivni upit s letovima, konkretno definiciju relacije `Letovi_presjedanje`. Graf ovisnosti bi u ovom slučaju imao jedan vrh imena `Letovi_presjedanje` i samo jedan brid, što je vidljivo na slici 3.6a. Taj brid nije označen oznakom '-' jer je upit koji definira `Letovi_presjedanje` monoton u odnosu na relaciju `Letovi_presjedanje`.



(a) upit s letovima



(b) upit sa subvencijama

Slika 3.6: Grafovi ovisnosti:

Nadalje, pretpostavimo da knjige iz tablice 1.1 imaju ocjenu i da će sve knjige čija je ocjena veća ili jednaka 4.0 biti subvencionirane sredstvima iz dva izvora: *Prvi* i *Drugi*. Dobro ocijenjene knjige mogu biti subvencionirane samo iz jednog izvora. Rekurzivni upit kojim bismo pokušali dobiti subvencionirane knjige po izvorima izgledao bi ovako:

[9] Biti definiran preko neke relacije R znači da se u definiciji rekurzivne relacije pojavljuje relacija R.

```
WITH
    RECURSIVE Prvi(ISBN) AS
        (SELECT ISBN FROM Knjige WHERE Ocjena >= 4.0
        AND ISBN NOT IN (SELECT ISBN FROM Drugi)),
    RECURSIVE Drugi(ISBN) AS
        (SELECT ISBN FROM Knjige WHERE Ocjena >= 4.0
        AND ISBN NOT IN (SELECT ISBN FROM Prvi))
```

Graf ovisnosti za ovaj primjer prikazan je na slici 3.6b. Vidimo da se u grafu nalazi ciklus koji sadrži čak dva označena brida, pa zaključujemo da ovaj rekurzivni upit nije dozvoljen u SQL3 standardu.

Osim spomenutih zahtjeva na rekurzivne upite, u literaturi se često analiziraju posljedice miješanja negacije i rekurzije, kao i načini na koji se ipak može osigurati izvođenje i korektnost kritičnih upita, poput slojevite negacije (eng. *stratified negation*), koristeći *stratum* u grafu ovisnosti (za više pogledati [24]).

Bibliografija

[1] *Information Technology - Database Language SQL*, 1992., https://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt, (studeni 2022.).

[2] *SQLite: The WITH Clause*, https://www.sqlite.org/lang_with.html, (studeni 2022.).

[3] https://github.com/antonelab/SQL.

[4] S. Abiteboul, R. Hull i V. Vianu, *Foundations of Databases*, Addison-Wesley Publishing Company, 1995.

[5] S. Abiteboul i V. Vianu, *Expressive Power of Query Languages*, 1993., https://www.sciencedirect.com/science/article/pii/B9780127082400500137, (studeni 2022.), str. 207–251.

[6] S. Adali, *Logic for Relational Databases and Relational Algebra for Bags*, 2016., http://www.cs.rpi.edu/~sibel/csci4380/spring2016/course_notes/logic_bagsemantics.html, (studeni 2022.).

[7] A. V. Aho i J. D. Ullman, *Universality of Data Retrieval Languages*, https://dl.acm.org/doi/pdf/10.1145/567752.567763, (studeni 2022.).

[8] M. Anderson, D. v. Melkebeek, N. Schweikardt i L. Segoufin, *Locality of queries definable in invariant first-order logic with arbitrary built-in predicates*, https://cs.union.edu/~andersm2/research/papers/avmss11.pdf, (studeni 2022.).

[9] T. Babić, *What is the Difference Between COUNT(*), COUNT(1), COUNT(column name), and COUNT(DISTINCT column name)?*, (2020.), https://learnsql.com/blog/difference-between-count-distinct/, (studeni 2022.).

[10] C. J. Date, *Logic and Databases: The Roots of Relational Theory*, Trafford Publishing, 2007.

[11] G. Dong, L. Libkin i L. Wong, *Local properties of query languages*, Theoretical Computer Science **239** (2000.), br. 2, 277–308.

[12] H. Garcia-Molina, J. D. Ullman i Widom J., *Database System - The Complete Book*, Pearson Prentice Hall, 2009.

[13] J. Gehrke, *Relational Algebra and SQL*, https://www.cs.cornell.edu/projects/btr/bioinformaticsschool/slides/gehrke.pdf, (studeni 2022.).

[14] L. Hella, L. Libkin, J. Nurmonen i L. Wong, *Logics with Aggregate Operators*, https://homepages.inf.ed.ac.uk/libkin/papers/lics99b.pdf, (studeni 2022.).

[15] A. Kozubek-Krycuń, *The History of SQL Standards*, https://learnsql.com/blog/history-of-sql-standards/, (studeni 2022.).

[16] L. Libkin, *Expressive power of SQL*, Theoretical Computer Science **296** (2003.), br. 3, 379–404.

[17] ———, *Locality of queries and transformations*, 2005., https://homepages.inf.ed.ac.uk/libkin/papers/wollic05.pdf, (studeni 2022.).

[18] ———, *Elements of Finite Model Theory*, Springer, 2012.

[19] L. Liu i T. M. Özsu, *Encyclopedia of Database Systems*, Springer, 2009.

[20] D. Lukichev, *Recursion in SQL Explained Visually*, (2020.), https://medium.com/swlh/recursion-in-sql-explained-graphically-679f6a0f143b, (studeni 2022.).

[21] R. Manger, *Baze podataka*, Element, 2014.

[22] J. Melton i A. R. Simon, *Understanding the new SQL: a complete guide*, Morgan Kaufmann Publishers, 1993.

[23] M. Vuković, *Matematička logika*, Element, 2009.

[24] J. Yang, *SQL3 Recursion*, 1999., http://infolab.stanford.edu/~ullman/fcdb/spr99/lec13.pdf, (studeni 2022.).

[25] C. G. Zarba, *Lecture Notes on Decision Procedures*, 2006., https://web.archive.org/web/20070929131504/http://react.cs.uni-sb.de/~zarba/snow/ch01.pdf, (studeni 2022.).

Sažetak

U ovome radu se bavimo proučavanjem izražajne snage jezika SQL, najviše u pogledu nemogućnosti iskazivanja rekurzivnih upita u jezgrenom dijelu SQL-a koji pokriva standard jezika iz 1992. godine. Granice izražajnosti postavljamo pomoću svojstva lokalnosti upita te dajemo skicu dokaza teorema o lokalnosti upita, koristeći pritom proširenje relacijske algebre agregatima, logiku s agregatima i logiku s brojanjem. Bavimo se važnim dodatkom jeziku SQL u standardu iz 1999. godine rekurzijom, te se osvrćemo na ograničenja vezana uz zadavanje rekurzivnih upita u tom standardu.

U prvom poglavlju definiramo pojmove vezane uz relacijski model baza podataka i operatora relacijske algebre. Na konkretnom primjeru baze podataka prikazujemo djelovanje definiranih operatora.

U drugom poglavlju opisujemo djelovanje operatora jezgrene verzije SQL-a koristeći pritom bazu podataka iz prvog poglavlja. Dajemo i kratak pregled povijesti SQL-a u pogledu standardizacije jezika. U zadnjem dijelu poglavlja bavimo se rekurzijom, motivacijom za uvođenje rekurzije i praktičnim primjenama rekurzivnih upita.

U trećem i završnom poglavlju uvodimo proširenje relacijske algebre agregatnim funkcijama, grupiranjem i aritmetičkim operacijama u oznaci $\mathrm{ALG}_{\mathrm{aggr}}$. Uvodimo pojam lokalnosti i iskazujemo rezultate o izražajnoj snazi SQL-a koristeći to svojstvo. U nastavku poglavlja dajemo skicu dokaza važnog rezultata (teorem 3.3.8), uvodeći pritom dva jezika za postavljanje upita bazirana na logici — logike $\mathcal{L}_{\mathrm{aggr}}$ i $\mathcal{L}_{\mathrm{C}}$. Na kraju poglavlja iskazujemo ograničenja vezana za zadavanje rekurzivnih upita te se bavimo svojstvima upita poput monotonosti, linearnosti, egzistencije i jedinstvenosti fiksne točke i slično.

Summary

In this paper, we deal with examining the expressive power of the SQL language, mostly in regards to the inability to express recursive queries in the core SQL language which encompasses the standard from 1992. Expressiveness boundaries are set using the feature of locality. We provide a draft of the proof of the locality of queries by using the extension of relational algebra via aggregates, aggregate logic and counting logic. We deal with an important addition to the SQL language in the standard from 1999 recursion, as well as consider the limits related to defining recursive queries in that standard.

In the first chapter, we define terms related to the relational database model and relational algebra operators. We showcase the effect of the defined operators by giving a practical example.

In the second chapter, we describe the effect of core SQL operators while using the database examples from the first chapter. We also provide a short overview of the history of SQL with regards to language standardization. In the last part of the chapter we deal with recursion, the motivation for introducing recursion and give practical examples of recursive queries.

In the third and final chapter, we introduce the extension of relational algebra via aggregate functions, grouping and arithmetic operations, denoted by $\mathrm{ALG}_{\mathrm{aggr}}$. We introduce the notion of locality and use it to state the results of the expressive power of SQL. Furthermore, we provide a sketch of the proof of an important result (theorem 3.3.8), while also introducing two query languages based on logic — logic $\mathcal{L}_{\mathrm{aggr}}$ and $\mathcal{L}_{\mathrm{C}}$. At the end of the chapter, we showcase the limitations related to defining recursive queries and also deal with query features such as monotonicity, linearity, the existence and uniqueness of a fixed point, etc.

Životopis

Rođena sam 21.12.1998. godine u Rijeci. Osnovnu školu Frana Krsto Frankopana u Vrhu na otoku Krku završavam 2013. godine. Iste godine upisujem matematički smjer Gimnazije Andrije Mohorovičića u Rijeci. Tijekom srednje i osnovne škole sudjelujem na brojnim natjecanjima, kao i šahovskim turnirima. Nakon završene mature 2017. godine upisujem Preddiplomski sveučilišni studij matematike na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu. Studij završavam 2020. godine i tada na istom fakultetu upisujem diplomski studij Matematika i računarstvo. Za vrijeme studija članica sam šahovske ekipe Fakulteta i Sveučilišta te sudjelujem na regionalnim, državnim i svjetskim natjecanjima, zbog čega dobivam nagradu Fakulteta za postignute iznimne rezultate u izvannastavnim aktivnostima. Također, suautorica sam članka na temu Predviđanje tipa osobnosti objavljenog u 40. broju Hrvatskog matematičkog elektroničkog časopisa math.e te osvajam 5. mjesto na ekipnom LUMEN Data Science natjecanju 2020. godine.