Prostorne baze podataka

Poljanić, Tea

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Tea Poljanić

PROSTORNE BAZE PODATAKA

Diplomski rad

Voditelj rada:
prof. dr. sc. Robert Manger

Zagreb, srpanj, 2021.

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. _____

Posvećujem ovaj rad svojim roditeljima kojima sam neizmjerno zahvalna na bezuvjetnoj ljubavi, na svakoj riječi utjehe i ohrabljenja te na razumijevanju i bodrenju u svakom trenutku. Hvala Vam što nikad niste sumnjali u mene!

Hvala mom bratu Vlatku i ostaloj obitelji na podršci i pomoći kroz svo ovo vrijeme.

Veliko hvala mojim prijateljima što su uljepšali ovaj period studiranja, ispunili ga srećom i zabavom i stvorili sa mnom trenutke za pamćenje.

Posebno hvala mojoj Luciji i Vinki koje su mi bile potpora od prvog dana, koje su prošle sa mnom sve vesele i one manje vesele studntske dane i bile oslonac kakvog sam samo mogla poželjeti.

Hvala Infoartu i svim divnim ljudima koji su me uveli u poslovni svijet te svojom pomoći i poticanjem olakšali pisanje ovog rada.

Na kraju, veliko hvala mentoru, prof. dr. sc. Robertu Mangeru, na prenesenom znanju te pomoći i savjetima za pisanje diplomskog rada.

Sadržaj

Uvod

Prostorne baze podataka su posebna vrsta baze podataka koja daje mogućnost spremanja prostornih podataka. U današnje vrijeme, upotreba takvih podataka je sve učestalija te je potreba za spremanjem prostornih podataka veća i veća. Zbog toga dolazi do razvoja prostornih baza podataka koje omogućuju lakšu pohranu te brži pristup podacima. U ovom diplomskom radu biti će objašnjen pojam prostorne baze podataka i opisane mogućnosti rada s takvom bazom.

Kako bi uopće mogli definirati prostorne podatke i prostorne baze podataka, potrebno je objasniti što su geoinformacijski sustavi, što se radi u prvom poglavlju ovog rada. Potom se prelazi na glavni dio rada, odnosno prostorne baze podataka. U drugom poglavlju biti će detaljnije objašnjeno što predstavljaju takve baze, kakvi se podaci mogu spremati u njih te kako indeksirati prostorne podatke kako bi dobili što efikasnije rezultate.

S obzirom da postoji više sustava za upravljanje prostornim bazama podataka, u ovom radu odabran je PostgreSQL i njegovo prostorno proširenje PostGIS. U trećem poglavlju obradit će se PostgreSQL i PostGIS te prikazati kako se pomoću konkretnog softvera prikazuju prostorni podaci te koje funkcije postoje za rad s njima.

Na kraju, kao studijski primjer ovog rada, prikazat će se stvarna upotreba prostornih baza podataka na primjeru PayDo aplikacije. Cilj je unaprijediti aplikaciju za plaćanje parkinga tako da se pomoću prostorne baze podataka omogući brže pronalaženje zone parkiranja.

Poglavlje 1

Geoinformacijski sustavi

1.1 Općenito o geoinformacijskim sustavima

Pošto su geoinformacijski sustavi široko rasprostranjeni te se koriste u raznim tehnologijama, postoje i mnoge definicije geoinformacijskih sustava. Općenita definicija bi bila da je to alat za izradu prostornih informacija i njihovu upotrebu. Nešto detaljnija definicija bi bila sljedeća: geoinformacijski sustav (eng. *Geographic Information System - GIS*) računalni je sustav dizajniran za pomoć u prikupljanju, održavanju, skladištenju, analizi i distribuciji prostornih podataka. Kao što samo ime govori, GIS objedinjuje geografiju i informatiku, ali i razne druge znanosti poput matematike i kartografije. Zato se GIS smatra i "spremnikom" karata u digitalnom obliku. Međutim, GIS je puno više od karata. Pomoću njega može se proširiti područje prikaza koje je inače ograničeno dimenzijama karte. Također, osim 2D prikazivanja kao na kartama, moguće je i 3D prikazivanje. Zbog toga GIS možemo smatrati pametnim kartama.

Glavni podsustavi geoinformacijskih sustava su:

1. Sustav za unos podataka
   Ovaj podsustav bavi se prikupljanjem i obrađivanjem prostornih podataka iz već postojećih karata, daljinskih senzora, mapa i slično.

2. Sustav za pohranu i dohvat podataka
   Ovo je podsustav koji organizira pohranjene podatke tako da pristup njima zbog naknadnih analiza bude što brži. Također dopušta brza i točna ažuriranja unesenih podataka.

3. Sustav za analizu podataka
   Omogućuje analizu podataka te izvodi zadatke kao što je promjena strukure podataka.

4. Sustav za izvještavanje
   Prikazuje sve podatke iz baze podataka ili neki određeni dio dobiven manipulacijom podataka. Stvaranje ovih prikaza zapravo uključuje sve ono što se smatra računalnom kartografijom. To područje predstavlja proširenje tradicionalne kartografije.

Također, možemo definirati i četiri komponente GIS sustava bez kojih sustav ne može funkcionirati. To bi bile sljedeće komponente: hardver, softver, podaci i ljudi.

- Hardver: obuhvaća svu opremu koja je potrebna za većinu aktivnosti u GIS-u (npr. računalo, skeneri i pisači, mrežni uređaj)
- Softver: alati koji se koriste za kreiranje, promjenu, prikaz i analizu podataka
- Podaci: nalaze se u bazama podataka, a dijele se na: prostorne (geografske) i neprostorne
- Ljudi: stručnjaci koji su osposobljeni za upravljanje i održavanje sustava

1.2 Povijest geoinformacijskih sustava

Prva pojava povezivanja događaja s mjestima događa se 1854. godine u doba izbijanja kolere. U početku se vjerovalo da se bolest širi zrakom. Međutim, doktor John Snow nije vjerovao u to pa je odlučio mapirati događaje. Bilježio je mjesta izbijanja epidemije, ceste, granice imanja i pumpe za vodu. Pomoću toga došao je do obrasca pomoću kojeg je dokazao da se bolest ne prenosi zrakom nego vodom. Osim toga, došao je do zaključka da se bolest prenosila samo pomoću jedne pumpe. To je označavalo početak novog proučavanja širenja bolesti te je bio i početak prostorne analize.

Narednih godina nije bilo velikog napretka u razvijanju jer se mapiranje tj. prikazivanje podataka na geografskim kartama radilo na papiru. Tek od 1960. godine dolazi do poboljšanja. Napredak u računalnoj tehnologiji dovodi i do stvaranja geoinformacijskih sustava. Između 1960. i 1975. godine bila su tri glavna napretka: mogućnost printanja karata pomoću pisača, napredak u pohrani podataka te unapređenje procesorske snage računala. Postojala je mogućnost unosa koordinata kao podataka te računanje s njima. Posebno bitan u razvoju geoinformacijskih sustava smatra se Roger Tomlinson koji 1962. godine kreće s razvojem Kanadskog geoinformacijskog sustava (eng. *Canadian Geographis Information System - CGIS*). CGIS je bio prvi sustav koji je dopuštao preklapanje, mjerenje te slojevito rukovanje s kartama. Zbog toga se i Roger Tomlinson naziva i ocem GIS-a.

Jack Dangermord, koji se i suosnivač ESRI-a (*Environmental Systems Research Institute*), proučavao je znanost o okolišu, urbani dizajn i arhitekturu te je želio primjeniti računalno mapiranje u svojoj struci. Zbog toga se 1967. godine krenuo tim baviti na Harvardu u Laboratoriju za računalnu grafiku (*Harvard Lab for Computer Graphics*). Taj laboratorij je 1970-tih godina razvio prvi vektorski GIS pod imenom ODYSSEY GIS.

Krajem 70-tih dolazi do napretka u računalnoj memoriji i grafici. To je dovelo do stvaranja komercijalnog GIS softvera. Danas najveća svjetska GIS softverska tvrtka ESRI tada je bila jedan od dobavljača softvera. Oni su proizveli programski paket ARC/INFO, a danas je vodeća svjetska tvrtka za GIS softver.

Do najvećeg razvoja GIS-a dolazi između 1990. i 2010. godine. Pošto su računala postala jeftinija, ali brža i snažnija, dolazi do stvaranja sve više GIS softverskih opcija te su digitalizirani podaci za mapiranje postali lakše dostupni. Razvoju je znatno pridonijelo i lansiranje novih satelita te se pomoću njih razvilo sve više i više aplikacija te je GIS postajao sve popularniji. 1992. godine objavljena je prva digitalna karta svijeta te se to smatra početkom web GIS-a. Zbog povećanog usvajanja GIS-a, nastaje GIS s otvorenim kodom, Landsat satelitske snimke postaju dostupne svima te je GIS postao sveprisutan.

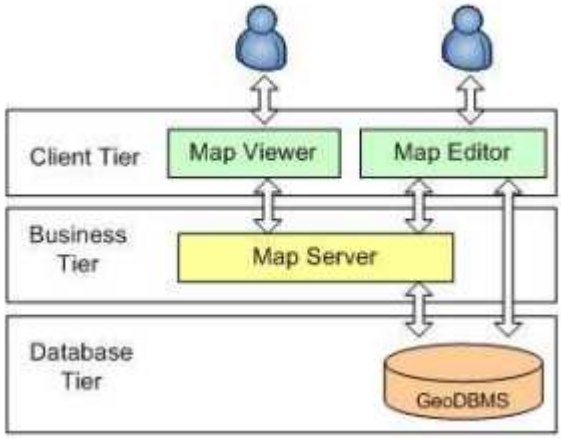
1.3 Arhitektura geoinformacijskih sustava

GIS softver ima troslojnu arhitekturu, takozvanu *Three-tier architecture*, koja se standardno koristi u modernim poslovnim aplikacijama. Na slici 1.1 nalazi se primjer takve strukture.

Arhitektura se sastoji od tri sloja: sloj s bazama podataka, poslovni sloj te korisnički sloj. Na razini baza podataka nalazi se sustav za upravljanje bazama podataka koji omogućuje pristup svim geoprostornim podacima pohranjenim u tablicama (GeoDBMS). U središnjem, odnosno poslovnom sloju, nalazi se Map Server koji objavljuje podatke koje je dohvatio iz baze. Također, isti sloj zapisuje podatke u bazu. Na kraju, nalazi se korisnički sloj koji sadrži Map Viewer i Map Editor. Map Viewer služi samo za čitanje podataka s poslužitelja, dok Map Editor omogućuje izmjenu podataka.

GeoDBMS je sustav za upravljanje bazom podataka koji podržava pohranu i pretraživanje geografskih podataka. Uobičajeno, takav se sustav stvara tako da se proširi već postojeći relacijski DBMS (primjerice Oracle, PostgreSQL, MySQL) s posebnom skupinom geografskih podataka i funkcija. Neka od takvih proširenja su Oracle Spatial, PostGIS te ESRI ArcSDE.

Map Server softverska je komponenta koja se obično izvodi unutar web poslužitelja (npr. Apache HTTP poslužitelj). Njezin cilj je klijentima pružiti dinamički generirane



Slika 1.1: Troslojna arhitektura GIS-a

mape iz dobivenih podataka. Te mape su zapravo u formatu slikovne datoteke koja se može prikazati na zaslonu računala.

Map Viewer je klijentska aplikacija koja prikazuje karte na korisničkom sučelju tako da se mogu pregledavati i analizirati. Map Editor dodatno proširuje funkcijonalnost Map Viewer-a dodavanjem mogućnosti stvaranja novih slojeva, izmjene značajki, izvođenja napradne geo-obrade i slično. Viewer se obično temelji na Internet aplikaciji koja zahtjeva mape od Map Servera koristeći neki protokol, dok je Editor desktop aplikacija koja također komunicira s poslužiteljima mapa ili direktno s bazom podataka.

Poglavlje 2

Prostorne baze podataka

Kako bi mogli definirati što su to prostorne baze podataka, najprije ćemo reći nešto općenito o bazama podataka. Baza podataka je skup međusobno povezanih podataka koji su pohranjeni u vanjskom memoriji računala. Omogućeno je ubacivanje novih podataka, promjena i brisanje starih te čitanje postojećih podataka.
Sustav za upravljanje bazom podataka - SUBP (eng. *Data Base Management System - DBMS*) je poslužitelj baze podataka. Sve izmjene na bazi obavljaju se posredovanjem SUBP-a. On obavlja sve operacije s podacima u ime klijenta te brine za sigurnost podataka.
Bazu podataka možemo smatrati hijerarhijom apstrakcija gdje je svaka razina jedna vrsta modela (prikazuje skup objekata i operacija nad objektima). Zadatak SUBP-a je omogućiti računalnu realizaciju svakog modela. Na jednoj razini hijerarhije nalazi se model podataka, a o njemu ćemo se detaljnije baviti kasnije u ovom radu.

2.1 Prostorni podaci

Kroz promatranje samog GIS-a, vidimo da je GIS sustav za obradu prostornih podataka. Iz samog naziva se vidi da su to podaci povezani s prostorom. Prostorni podaci su zapravo podaci koji se mogu prikazati i analizirati pomoću atributa koji označavaju mjesto na površini Zemlje ili u njenoj blizini. Obično se takav atribut prikazuje pomoću kooridanata koje definiraju položaj i oblik određenog "tijela" koje treba mijeriti i grafički prikazati.
Prostorni podaci imaju dva važna svojstva:

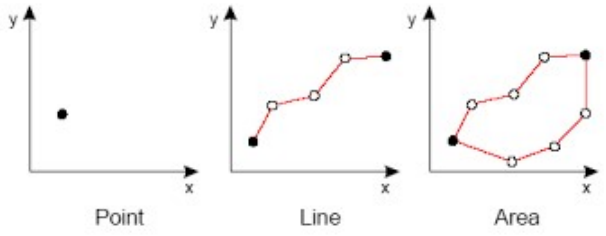
- Oni su referenca na geografski prostor, što bi značilo da predstavljaju prihvaćeni koordinatni sustav koji obuhvaća neko područje Zemlje. Zbog toga se mogu kombinirati podaci iz više izvora.

- Podaci se bilježe u relativno malim zemljopisnim mjerilima te tako predstavljaju veliko područje na Zemlji ili oko nje.

Prostorne podatke možemo podijeliti u dva temeljna oblika: vektor i raster.

Vektorski podaci

Vektorski podaci sastoje se od točaka, linija i poligona. Najjednostavnije, vektorski podaci sastoje se od pojedinačnih točaka koje su pohranjene kao koordinatni parovi i tako prikazuju mjesto u stvarnom svijetu. Spajanjem tih točaka određenim redosljedom stvaraju se linije ili, ako se zatvore, poligoni (kao što se može vidit na slici 2.1). Točka reprezentira objekt za koji je bitno znati položaj u prostoru, ali ne i njegova veličina. Linija, tj. niz povezanih linija, je apstrakcija za niz povezanih objekata u prostoru, dok je poligon apstrakcija za objekte koji se prostiru u dvodimenzionalnom prostoru. Takvi podaci korisni su za pohranjivanje i predstavljanje podataka koji imaju diskretne granice, poput državnih granica, ulica, granica parcela i slično. Poznat primjer korištenja vektorskih podataka je Google Maps.
Vektorski podaci smatraju se tradicionalnom metodom za kartografsko prikazivanje. Takav prikaz vizualno je ugodniji. Spremanjem podataka ne gube se nikakvi podaci te se čuvaju točne informacije. U atribute vektora se također mogu pohraniti dodatni atributi i polja podataka. Međutim, obrada i pohrana takvih podataka može bit teža. Pošto se vektorski podaci pohranjuju kao niz točaka, potrebno je svaku stavku posebno pohraniti. Također, svaka točka ili skup točaka može imati pridružene podatke koji mogu produžiti vrijeme pohrane i obrade. S više značajki za točke ili više točaka, stvaraju se dulji procesi te algoritmi postaju složeniji.

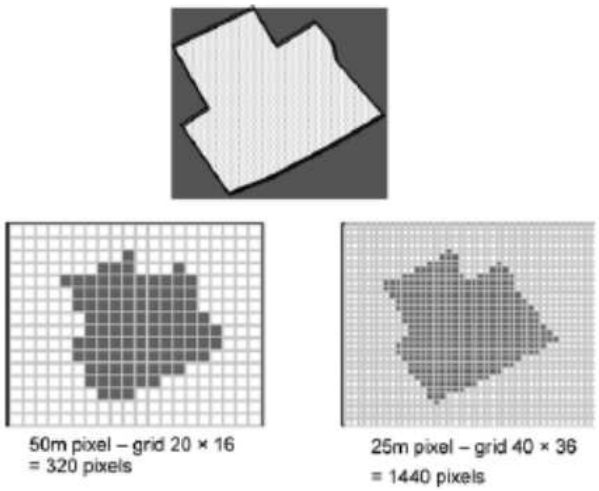


Slika 2.1: Primjeri vektorskih podataka

Rasterski podaci

Rasterski podaci prikazuju svijet kao površinu podijeljenu na manje ćelije, pri čemu svaka od tih ćelija ima pridruženu neku vrijednost. To možemo usporediti sa digitalnom fotogra-

fijom. Svaki piksel je zapravo jedna ćelija koja odgovara određenoj vijednosti boje. Kada ćelije prebacimo u GIS, one mogu sadržavati i druge podatke, primjerice temperaturu ili oborine. Glavna razlika između digitalne fotografije i GIS-a je ta što GIS sadrži i dodatne podatke koji detaljno opisuju koliko ćelije mogu biti velike. Mala veličina odnosno rezolucija daje bolji prostorni prikaz, ali zato rezultira puno većom datotekom podataka. Razliku možemo vidjeti na slici 2.2.



Slika 2.2: Prikaz različite rezolucije rastera

Rasterski podaci prikladniji su za matematičko modeliranje i analize. Zbog činjenice da rasterske površine (ćelije) predstavljaju jedan atribut, izračuni i algoritmi mogu biti brzi i jednostavni za izvođenje. Ovakav prikaz idealan je za prikazivanje i pohranjivanje kontinuiranih vrijednosti. Uz to, pohrana rastera može biti mala zbog načina na koji se pohranjuje njihov zemljopisni položaj. Koordinate svake ćelije prikazuju se njezinim položajem u mreži. Potrebno je znati početnu točku, primjerice gornji lijevi kut, veličinu ćelije te broj redaka i stupaca. To će postaviti mrežu na projekciju karte i pružiti točnu veličinu i prikaz. Jedan od glavnih nedostataka je veličina ćelije. Veličina zapravo označava razlučivost slike. Veličina ćelije može odrediti veličinu površine za pohranu i mjerilo koje se koristi. Veličina ćelije od kilometra može biti dobra za rad na državnoj razini, ali na lokalnoj to ne bi funkcioniralo.
Rasterski podaci više odgovaraju kontinuiranim prikazima, zapravo mogu označavati samo jednu karakteristiku. S manjim mrežnim ćelijama dobili bi detaljniju površinu, ali bi spremište podataka bilo veće te obrada može biti dulja. Još jedan od nedostataka je taj što se većina podataka na početku pohranjuje kao vektorski. Kako bi se moglo s njima

raditi u rasterskom formatu, potrebno je pretvoriti podatke. Tada može doći do problema ako se koriste neprikladne veličine ćelija koje pojednostavljuju podatke.

2.2 Prostorne baze podataka

Sada kada imamo definiciju baze podataka i SUBP-a te smo definirali koji prostorni podaci postoje, potrebno je definirati baze u koje se oni spremaju.

Prostorna baza podataka je baza podataka koja definira posebne vrste podataka za geometrijske objekte i omogućuje njihovo pohranjivanje u standardne tablice baza podataka. Osim pohranjivanja, nudi posebne funkcije i indeksiranje za postavljanje upita na bazu koristeći upite nalik SQL-ovim. Prostorne baze podataka nude alate za pohranu i analizu. Osim što može vizualno prikazivati podatke, pomoću nje se mogu napraviti brojne analize podataka. Takve analize ne moraju biti ograničene brojem varijabili s kojima se treba raditi. Također, mogu koristiti podatke poput točaka, linija ili poligona koje uvelike mogu olakšati pretraživanje.

Kao što je već navedeno, 2D mapiranje može se postići samo sa točkama, linijama i poligonima. Pomoću njih se mogu modelirati fizičke zemljopisne cjeline. Na primjer, autocestu možemo modelirati pomoću tih vrsta podataka. Dva grada između kojih se proteže cesta mogu biti poligoni, cestu definiramo pomoću linija, dok, primjerice, benzinsku postaju možemo prikazati pomoću točke.

Možemo sada definirati i sustav za upravljanje geoprostornim bazama podataka (SUGBP). SUGBP je softverski modul, implementiran proširenjem objektno-relacijskog ili objektno-orijentiranog sustava baze podataka. Posjeduje prostorne apstraktne tipove podataka, kao i jezik pomoću kojeg se stvaraju upiti na takve podatke. SUGBP podupire prostorno indeksiranje te specifična pravila za optimizaciju upita na prostorne baze podataka, kao i algoritme za operacije definirane na prostornim tipovima podataka.

Kako bi se davali odgovori na upite, baze uvode nove tipove podataka koji pohranjuju geometrijske podatke. Takvi tipovi podataka ravnopravni su s već postojećim tipovima u standardnim bazama te se mogu koristiti za pohranu podataka u tablicama. Kako bi se mogli izvršavati upiti nad njima, potrebno je definirati prostorne operatore koji rade s takvim tipom podataka. Obično oni budu realizirani kao funkcije. Struktura podataka, njihova pohrana te operatori nad njima standardizirani su kroz *Open Geospatial Consortium*, u kojem su zastupljeni svi vodeći proizvođači SUBP-a, gdje je definiran standard za uključivanje dvodimenzionalnih podataka u SQL. Kasnije u ovom radu definirat ćemo geometrijske tipove koje ima implementiran PostgreSQL.

2.3 Modeli prostornih baza podataka

Prije kreiranja baze podataka, potrebno je odrediti koje će tablice ona sadržavati, pojedini atribut za svaku tablicu i veze među tablicama. Takvo definiranje baze podataka zovemo shemom baze podataka. Za definiranje iste te sheme koristimo jezik koji se zove model podataka. U nastavku ćemo definirati modele koji se najčešće koriste u prostornim bazama podataka, a to su: relacijski model, objektni model, objektno-relacijski model i polustrukturirani model. Tradicionalno se u bazama podataka koristi relacijski model, ali kod takvog modela postoje ograničenja i nedostatci što se tiče definiranja kompleksnijih podataka pa tako i prostornih.

Relacijski model

Jedan od najraširenijih i napoznatijih modela baza podataka je relacijski model. Ovaj model zahtjeva da se baza sastoji od tablica tj. relacija te da svaka ima jedinstveno ime po kojoj se razlikujemo u bazi. Stupac te tablice izjednačavamo s atributom koji također ima svoje ime. Takav atrbut u nekim slučajevima ne treba biti unesen. Redak takve tablice predstavlja entitet i u tablici ne mogu postojati dva ista entiteta. Redak također nazivamo n-torka, a broj tih n-torki u tablici je kardinalnost relacije. Za jedan od atributa (stupaca) definiramo da je ključ pomoću kojeg se jednoznačno određuje n-torka. Ako se ne može odrediti jedan takav, određuje se složeni ključ koji je kombinacija više atributa. Ukoliko postoji više kandidata za ključ, definira se primarni ključ. Tablice se mogu povezati pomoću stupaca koji se nalaze u obje tablice te je važno da ti stupci predstavljaju jedinstveni podatak.

Objektni model

Objektni model nastaje kao odgovor na probleme koji se događaju kod relacijskog modela. S obzirom da relacijski model ne koristi složene tipove podataka, stvara se objektni model. U takvom modelu u bazu se pohranjuju objekti. Objektni pristup nudi fleksibilnost jer ne sadrži samo osnovne tipove podataka nego nudi mogućnost stvaranja kompleksnijih objekata. Model u bazi podataka ne razlikuje se od modela u aplikaciji. Za upravaljanje takvim bazama podataka postoji sustav za upravljanje bazama podataka koji implementira objektno orijentirani model podataka (OOSUBP). Implementacija takvog sustava ovisi o samom programskom jeziku i uvelike se međusobno razlikuju. Godine 1991. utemeljen je konzorcij ODMG (*Object Database Management Group*) radi razvoja skupa standarda koje mora zadovoljavati OOSUBP. ODMG uključuje i objektni definicijski jezik (eng. *Object Definition Language - ODL*) te objektni upitni jezik (eng. *Object Query Language - OQL*). Također su definirani standardi za povezivanje s jezicima Java, Smalltalk te C++.

Objekt u ovakvom modelu mora imati identitet (eng. *object identity - OID*) koji je jedinstven (dva ili više objekata ne mogu imati isti OID niti jedan objekt može imati više

OID-a). Glavna značajka OID-a je da se ne mijenja te je poželjno, ukoliko se objekt obriše, da se njegov OID više ne koristi. Svaki objekt ima atribute i metode koje su implementirane za njega. Oni se nasljeđuju od nadređene klase, dok podklasa može definirati i nove.

Objektno-relacijski model

Objektno-relacijski model predstavlja kombinaciju jednostavnosti relacijskog modela sa nekim naprednijim funkcionalnostima objektnog modela. Ideja ovakvog modela je upravljanje objektima uz kompatibilnost sa relacijama. Zapravno ovaj model proširuje relacijski model objektnim konceptima (apstraktni tipovi podataka, učahurenje, nasljeđivanje i slično). Koncept relacijskog modela je temeljni koncept ovog modela, ali je proširen složenijim (korisničkim) tipovima podataka. Može se kreirati proizvoljan broj novih tipova podataka i rabiti ih isto kao osnovne tipove. Takvi tipovi podataka su učahureni (njihova implementacija nije vidljiva korisnicima te im se pristupa kroz određenu formu sučelja). Time se omogućuje promjena implementacije metode bez utjecaja na aplikaciju (podrazumijeva se da se semantika metode ne mijenja).

Polustrukturirani model

Kao što smo vidjeli u prethodnim modelima, struktura podataka je eksplicitno definirana u shemi baze podataka. To u polustruktuiranom modelu nije slučaj. U ovakvom modelu shema je sadržana u samim podacima. Opis strukture nije potreban, kao ni definiranje podataka. Neki objekti mogu imati više istih atributa dok drugi uopće ne moraju imati taj isti atribut. Također, atributi ne moraju biti istog tipa podataka. Podaci se mogu prikazivati putem stabala. Korijen tog stabla je objekt a listovi predstavljaju vrijednosti atributa tog objekta.

2.4 Prostorni upiti i indeksiranje

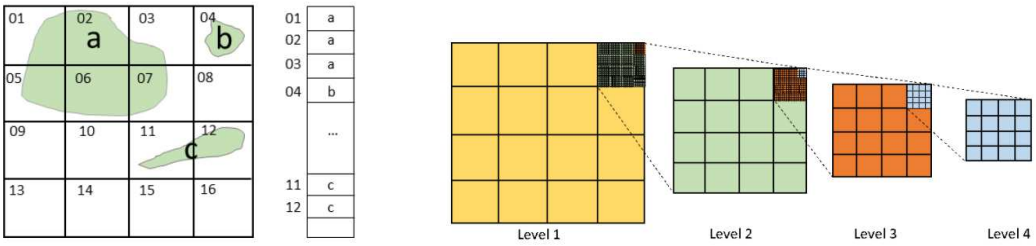
Kada imamo podatke u prostornim bazama, tada radimo upite. Prostorni upiti su upiti na bazu podataka koji koriste geometrijske funkcije za odgovaranje na pitanja o prostoru i objektima u prostoru. Standardnom SQL jeziku dodaju se funkcije koje rade s geometrijskim objektima u bazi podataka. Takav način rada sličan je onome koji radi s datumima. Kao što za vremenske podatke postoje funkcije koje određuju koliki je vremenski period između dva datuma, tako za prostorne podatke postoje npr. funkcije koje određuju udaljenost dvije točke.
Osim što takvi upiti odgovaraju na pitanja o prostoru, omogućuju izmjenu postojećih objekata i stvaranje novih. Taj dio prostorne analize naziva se geometrijskom ili prostornom

obradom.

Kako bi se upiti na prostornim bazama izvodili brže, definiraju se prostorni indeksi. Prostorni indeks je struktura koja omogućuje brži i učinkovitiji pristup nekom prostornom objektu. Bez indeksiranja, svako traženje neke značajke bi zahtijevalo čitanje svakog zapisa u bazi, što bi rezultiralo puno duljim vremenom obrade. Postoje razne vrste prostornih indeksa te s različitim indeksima se dobiju mjerljive razlike u vremenima izvođenja. Sada ćemo opisati neke vrste indeksiranja. Najprije opisujemo vrste indeksiranja u kojima se indeksira položaj objekta u prostoru pa zatim indeksiranje položaja objekta u bazi.

Indeksiranje pomoću fiksne mreže

Indeksiranje pomoću fiksne mreže čini $n \times n$ velik niz ćelija jednake veličine. Svaka ćelija je povezana s popisom prostornih objekata koji se sijeku ili preklapaju sa ćelijom. Primjer možemo vidjeti na slici 2.3a.



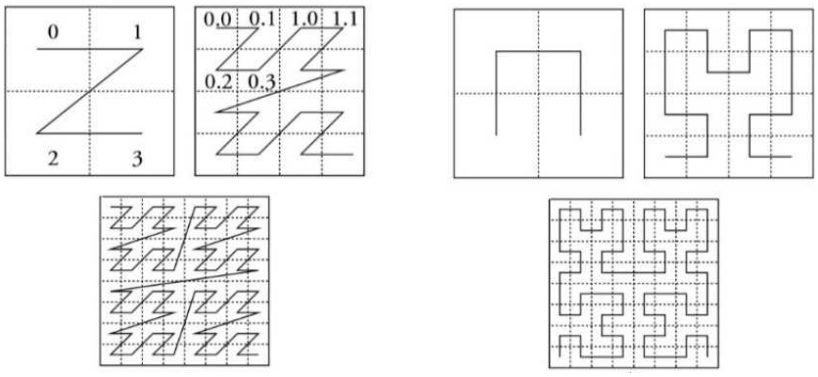
(a) Indeksiranje pomoću fiksne mreže

(b) Indeksiranje pomoću fiksne mreže razine 4

Slika 2.3: Prikaz indeksiranja

Ravnina se može podijeliti u mrežu po razinama. Što je razina veća, to je i razgradnja složenija. Na slici 2.3b možemo vidjeti razgradnju do razine 4 u 4×4 mrežu.

Kao što vidimo, ovakvo indeksiranje je napravljeno u dvodimenzionalnom prostoru. Problem nastaje kada to treba pohraniti jer se pohranjuje jednodimenzionalno. Potrebno je odrediti kojim redoslijedom će se ćelije u mreži spremati. To preslikavanje iz dvodimenzionalnog prostora u jednodimenzionalni treba biti takvo da oni elementi koji su bliži u prostoru budu bliži i na liniji te da se dva različita elementa nikada ne preslikaju u isti element na liniji. Dva najpoznatija primjera krivulja za linearizaciju višedimenzionalnog prostora su Z-krivulja te Hilbertova krivulja koje se mogu vidjeti na slici 2.4.



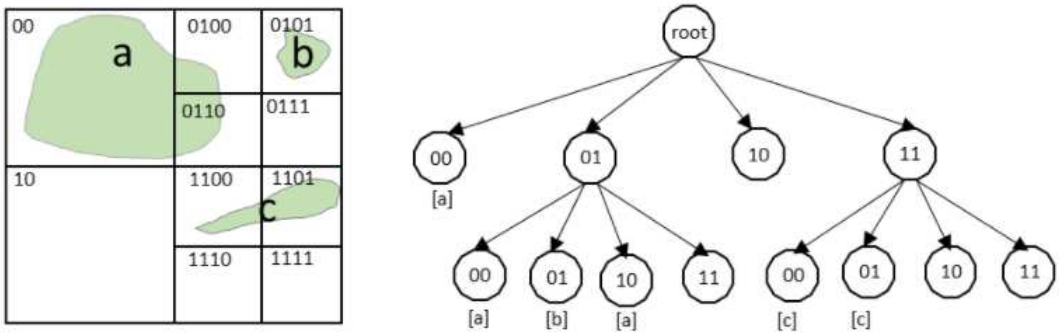
(a) Z-krivulja (b) Hilbertova krivulja

Slika 2.4: Krivulje za linearizaciju višedimenzionalnog prostora

Obilazak ćelija pomoću Z-krivulje radi se na sljedeći način: mreža se naprije podijeli na 4 kvadranta te se dodjele brojevi koji se povezuju po veličini. Isto se radi za svaki kvadrant, s tim da se naprije povezuju kvadranti na najnižoj razini pa prema višoj. Primjer takvoj povezivanja je na slici 2.4a. Problem takvog povezivanja je što se događaju ”skokovi” kada dolazi do prijelaza s jednog kvadranta na drugi. To se ne događa prilikom kreiranja Hilbertove krivulje. Kada se kreira takva krivulja, imamo definiranu nultu i prvu itaraciju kao što je prikazano na gornje dvije sličice na slici 2.4b. Kada to imamo, možemo graditi krivulju dalje. Druga iteracija se tvori tako da se u prvoj iteraciji pronađe segment sličan krivulji iz nulte iteracija i zamijeni se cijelom prvom iteracijom. Pomoću takvog načina kreiranja dobije se Hilbertova krivulja. Ovakve krivulje su beskonačno guste te nakon određenog broja iteracija potpuno prekrivaju prostor u kojem se nalaze.

Indeksiranje pomoću kvadrantnog stabla

Indeksiranje pomoću kvadrantnog stabla (eng. *quadtree*) popularna je tehnika prostornog indeksiranja. To bi zapravo bio oblik mreže koja se dijeli u skladu s gustoćom objekata koji su u njoj (slika 2.5).

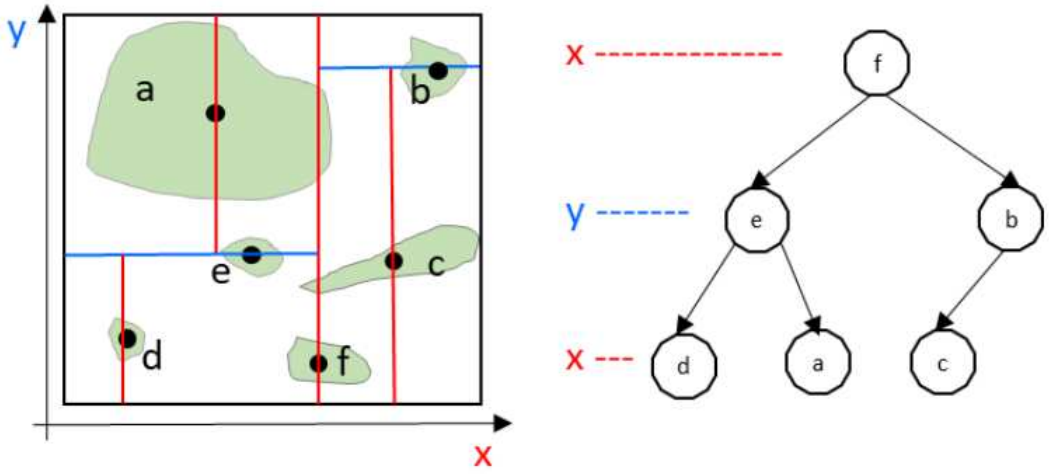


Slika 2.5: Struktura indeksiranja pomoću stabla četverokuta

Početna slika dijeli se na kvadrante sve dok se u jednom kvadrantu ne nalazi samo jedan objekt. Sve dok u kvadrantu postoji više objekata, on se dijeli na nove kvadrante. Takva slika se potom prikazuje pomoću stabla. Svaki kvadrant prikazan je pomoću čvora stabla. Čvor koji nije list predstavlja kvadrant koji ima više objekata u sebi i može se dalje dijeliti te je on točka daljnjeg grananja. Oni čvorovi koji su također listovi stabla predstavljaju kvadrante koji imaju samo jedan objekt u sebi i više se ne dijele. Dubina stabla ovisi o složenosti slike.

K-D stablo

K-D stablo ili K-dimenzionalno stablo je poseban slučaj binarnog stabla pretraživanja. To je struktura podataka za podjelu prostora i organiziranje objekata u k-dimenzionalnom prostoru. Za lakše razumijevanje, promatrat ćemo prostor dimenzije dva. Svaki čvor dijeli ravninu na dva dijela. Lijeva podravnina sadrži točke koje predstavljaju lijevo podstablo tog čvora, a desna točke koje predstavljaju desno podstablo (slika 2.6).



Slika 2.6: Prikaz strukture K-D stabla

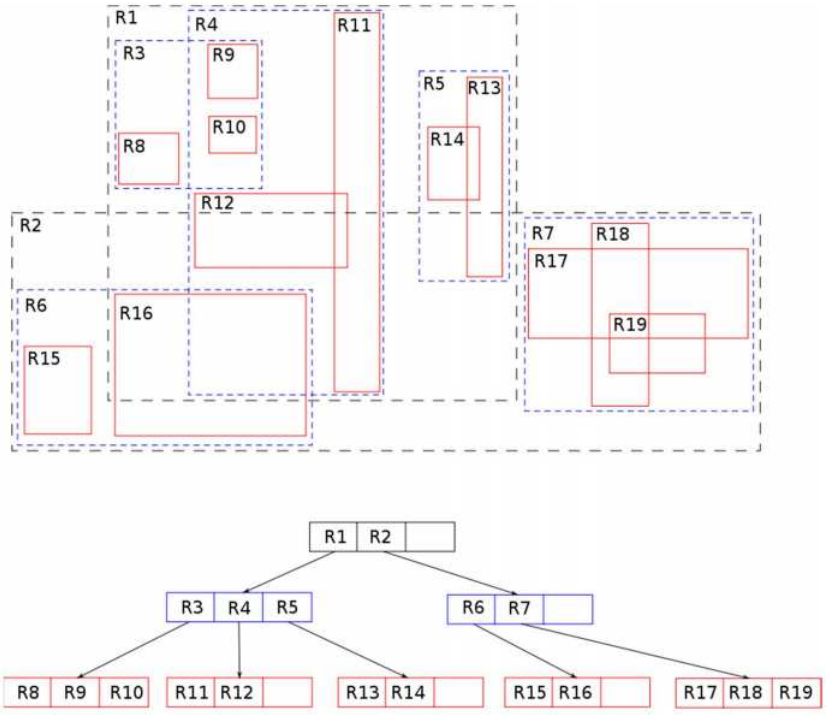
K-D stablo generira se tako da se prvo odabere jedan pravac za podjelu. Pravac se bira tako da dijeli odabranu os. Prva os koju dijelimo u našem slučaju je os x. Točka kroz koju prolazi pravac odabire se kao medijan koordinata točaka s obzirom na os koju trenutno dijelimo (za os x gledamo sve x koordinate naših točaka). Nakon što smo podijelili os x, prebacujemo se na os y i za svaka dva dobivena dijela radimo isto što u prethodnom koraku, samo gledajući os y. Postupak ponavljamo dok ne prođemo sve točke te smo tada izgenerirali K-D stablo. Stablo koje dobijemo je izbalansirano, svaki list je na istoj udaljenosti od korijena.

Prethodno navedena indeksiranja su bila indeksiranja objekata u prostoru. Sada navodimo indeksiranje pomoću položaja objekta u bazi.

R-stablo

Prije nego što definiramo R-stablo, potrebno je opisati B-stablo jer je R-stablo vrlo slično njemu. Iako je naziv sličan binarnim stablima, B-stabla se od njih razlikuju u nekoliko važnih karakteristika. Jedna od ključnih razlika je ta što u B-stablima čvorovi mogu sadržavati više vrijednosti ključeva te samim time pokazivati prema većem broju elemenata (u binarnom svaki čvor pokazuje na dva elementa). Ukoliko gledamo pretraživanje podataka u B-stablu, slično je onom u binarnom samo što odluku o prelasku u idući čvor treba donijeti s obrzirom na veći broj elemenata. Taj algoritam je složeniji nego onaj za samo dva elementa, ali upravo zbog toga ukupno vrijeme pretraživanja je kraće jer zahtjeva prolazak manje razina nego kod binarnog stabla. Dodavanje i brisanje elemenata također

je jednostavnije jer često ne dolazi do prestrukturiranja podataka po cijelom stablu.
Sada kada znamo kako je definirano B-stablo, možemo definirati i R-stablo.
R-stablo je zapravo slično B-stablu te se koristi za indeksiranje višedimenzionalinih informacija, primjerica koordinatnih parova. R-stablo je struktura podataka koja se koristi za prostorno pretraživanje. Takvu strukturu podataka predložio je Antonin Guttman te navodi da se R-stablo pokazalo kao optimala struktura za prostorne podatke. Svaki objekt u R-stablu prikazan je pomoću minimalnog ograničavajućeg pravokutnika (eng. *minimum bounding rectangle / box - MBR*). Čvorovi koji su listovi sadrže identifikator objekta i MBR unutar kojeg se objekt nalazi. Listovi su grupirani u veće pravokutnike koji čine unutarnje čvorove stabla. Osim što unutarnji čvorovi sadrže MBR-ove unutar kojih se nalaze dijeca, sadrže i pokazivače na svoju djecu. Također, čvorovi mogu sadržavati više elemenata ali samo do unaprijed određenog maksimalnog broja. Proces takvog grupiranja nastavlja se do posljednje skupine čvorova koji čine korijen stabla. Korijen tada predstavlja MBR koji sadrži sve objekte, a svaki čvor predstavlja MBR koji sadrži svoju djecu. Prilikom pretraživanja minimalni ograničavajući pravokutnici se koriste da bi odredili koji će se čvorovi dalje pretraživati. Tim postupkom se značajno smanjuje broj čvorova koje treba pretražiti.



Slika 2.7: Prikaz strukture R-stabla

Gledajući primjer na slici 2.7, ako želimo saznati presjecaju li se objekti R9 i R10, moramo najprije proći kroz R1. R1 sadrži pravokutnike R3, R4 i R5. Za svakog provjeravamo sadrže li R9 i R10. Pošto R3 sadrži oba pravokutnika, provjeravamo njihove geometrije te zaključujemo da se njihovi pravokutnici ne presjecaju. U slučaju da smo željeli provjeriti presjecaju li se R8 i R18, mogli smo stati već u prvom koraku zato jer se ne nalaze u istom MBR-u. Bez ovakvog indeksiranja bilo bi potrebno sekvencijalno pretražiti svaki element u bazi, što rezultira puno dužim izvršavanjem pretraživanja.

Za umetanje objekata u R-stablo potrebno je pronaći mjesto gdje ubaciti objekt. Potraga počinje od korijena stabla te se rekurzivno nastavlja dalje. Provjeravaju se svi pravokutnici u onom čvoru u kojem se nalazimo te se odabire onaj pravokutnik kojem treba najmanje uvećanje da obuhvati novi objekt. Ako postoji više takvih, odabire se onaj s najmanjom površinom. To se radi sve dok se ne dođe do lista. Ako nisu svi listovi puni, umeće se novi. Ako jesu, onda se list dijeli prije umetanja.

Ovako definirano R-stablo ima neke nedostatke. To su:

- Izvršavanje upita o lokaciji neke točke pomoću R-stabla može dovesti do potrage po nekoliko grana iz korijena. Takvo pretraživanje može dovesti do pogoršanja performansi, posebno ako je preklapanje MBR-ova značajnije.
- Nekoliko velikih pravokutnika može značajno povećati stupanj preklapanja što također dovodi do pogoršanja performansi tijekom izvršavanja upita zbog velikog praznog prostora

Zbog ovih nedostataka napravljene su sljedeće varijante R-stabala:

1. R^+-stablo
   Ovakva struktura napravljena je kako bi se izbjeglo posjećivanje više grana tokom upita o lokaciji točke. Time se dobiju poboljšane performanse pretraživanja. R^+-stablo izbjegava preklapanje MBR-ova na istoj razini stabla pomoću tehnika izrezivanja. Upiti koji traže raspon rade po sličnom algoritmu kao za R-stablo samo što ne postoje pojavljivanje istog predmeta u više MBR-ova.
2. R^*-stablo
   Kao što smo već vidjeli, R-stablo se temelji samo na minimizaciji MBR-ova. S druge strane, R^*-stablo radi sa još nekoliko kriterija prilikom kreiranja, od kojih se očekuje poboljšanje performansi. To bi bili sljedeći kriteriji:
   - Minimizacija područja obuhvaćenog svakim MBR-om - ovo je kriterij po kojem radi i standardno R-stablo
   - Minimizacija preklapanje između MBR-ova - smanjenjem preklapanja dolazi do manjeg broja grana kroz koje upit prolazi

- Minimizacija opsega MBR-a - kvadrati imaju najmanji opseg u usporedbi s pravokutnicima iste površine, smanjenje opsega pomaže smanjenju praznog prostora
- Maksimizacija korištenja prostora za skladištenje

R*-stablo pronalazi najbolje moguće rješenje koje zadovoljava prethodno navedene kriterije.

Poglavlje 3

PostGIS

3.1 PostgreSQL

PostgreSQL je objektno-relacijski sustav za upravljanje bazama podataka otvorenog koda. PostgreSQL koristi i proširuje SQL s raznim značajkama koje sigurno pohranjuju i smanjuju opterećenja na radu s podacima. To je zapravo baza podataka koja se sastoji i od standardne relacijske baze podataka i objektne baze podataka koja podržava osnovne komponente bilo kojeg objektnog modela baze podataka. Također je model proširiv jer korisnici mogu definirati i svoje tipove podataka i funkcije.

PostgreSQL se razvio iz projekta Sveučilišta u Kaliforniji Berkeley pod imenom Ingres. Voditelj Ingresovog tima, Michael Stonebraker, napustio je Berkeley. Međutim, 1985. godine se vraća te započinje post-Ingres projekt. To je bio projekt koji je rješavao probleme sa suvremenim sustavima baza podataka. Taj projekt je zatim dobio ime Postgres te je imao cilj dodati značajke za potpunu podršku za sve vrste podataka. Andrew Yu i Jolly Chen nakon toga 1994. godine izrađuju projekt kojem dodaju interpreter za SQL. Puštaju ga pod imenom Postgres95 te postaje poznat kao open-source potomak izvornog Postgresa. Taj projekt je 1996. godine preimenovan u PostgreSQL kako bi se prikazala podrška prema SQL-u.

PostgreSQL podržava i geografske objekte te služi kao spremište geoprostornih podataka. To njegovo proširenje naziva se PostGIS o čemu ćemo pričati o nastavku.

3.2 PostGIS

PostGIS je besplatan open-source softer sukladan s OGC-om (*Open Geospatial Consortium*) koji se koristi kao proširenje PostgreSQL-a. PostGIS proširuje mogućnosti PostgreSQL-a tako da dodaje mogućnost upravljanja geoprostornim tipovima podataka. Dodaje nove tipove podataka za prostorne podatke te funkcije za upravljanje njima. Jezik PostGIS-a sličan je SQL-u i omogućuje jednostavno izvrašavanje upita i analize nad takvim podacima.

Povijest PostGIS-a

S obzirom da je PostgreSQL već u sebi imao geometrijske tipove podataka, činilo se nepotrebnim dodavati još prostornih mogućnosti. Međutim, ti geometrijski tipovi su bili previše ograničeni za GIS podatke i analizu. Izgrađeni su za akademska istraživanja te nisu bili prikladni za GIS. Odgovor na pitanje kako pohraniti prostorne podatke tada je nudio dokument *Simple Features for SQL* [8]. Rješenje je bilo sljedeće: najprije prebaciti geometrijske podatke u relacijski model te ih kasnije ponovno prebaciti u geometrijski model ili kreirati geometrijske objekte. Prvo rješenje pokazalo se sporim pa je bilo potrebno kreirati geometrijske objekte. Prva implementacija PostGIS geometrijskog tipa podataka ubrzala je učitavanje za 100 puta za razliku od relacijskog modela. Jedino što je preostalo za implementirati bili su indeksi koji bi omogućili upite na velikim skupovima podataka. Pokazalo se dobrim to što je PostgreSQL već imao primjere korištenja R-stabla. S takvim indeksima postignuto je pristupanje podacima u milisekundama. Komponente koje je sustav tada sadržavao bile su sljedeće:

- prostorni objekti i standardizirani tekst
- osnovne funkcije na prostornim objektima
- proširenje za uvoz i izvoz u Javu
- prostorni indeksi

Te komponente bile su dovoljne za prvu verziju PostGIS-a koja je postala dostupna 31. svibnja 2001. godine.

Nakon toga, dolazi do razvijanja Mapservera, aplikacije koja je omogućavala vizualizaciju podataka iz baze. Neke od funkcija bile su komplicirane za implementirati, primjerice operatori dodirivanja, presjeka, unije, razlike i slično. No, ubrzo je napravljen projekt pod nazivom *Geometry Engine, Open Source - GEOS* čija je biblioteka sadržavala algoritme potrebne za implementaciju *Simple Features for SQL* specifikacija. Tek nakon što se povezao sa GEOS-om, PostGIS je imao potpunu podršku za *Simple Features for SQL.*

Problem koji se kasnije pojavio bio je problem pohranjivanja podataka. Naime, baza podataka je bila formirana tako da se nije uzelo u obzir koliko prostora podaci zauzimaju. Za sve ulazne podatke bio bi dodijeljen prostor kao da su trodimenzionalni, iako bi bili dvodimenzionalni. Za male objekte kao što su točke, zauzimanje memorije kao da je trodimenzionalni objekt rezultiralo bi zauzimanjem 200% više memorije nego što je potrebno. Zbog toga je sustav za pohranu podataka bio najsporiji te ga je bilo potrebno ubrzati. Trebalo je smanjiti strukture podataka te je pokrenut novi ekspreriment pod nazivom *"light-weight geometry"*. Tim projektom smanjila se veličina izvornog zaglavlja i zahtjevanih dimenzija te se tada smanjilo i preopterećenje. Osim toga, ponovno je napravljeno indeksiranje za takvu vrstu podataka. Nakon dvije godine razvijanja eksperimenta, projekt je bio uvršten u PostGIS te su performanse sustava za velike baze podataka poboljšane.

U kasnijem razvoju su performanse bile u glavnom fokusu razvijanja i poboljšanja. Dodano je još GIS funkcija, poput onih za izgradnju poligona i linija. Nova ažuriranja rade na usklađivanju sa standardima pa se tako dodaje podrška za krivulje te funkcije navedene u ISO SQL/MM standardu (uključuje mnogo veći broj prostornih objekata te dodaje razne krivulje). Dodaje se prefiks ST_ na imena funkcija kako bi se podudarali sa SQL/MM standardom. Svake godine dolazi do nekoliko novih, ažuriranih verzija PostGIS-a čime on sve više i više napreduje.

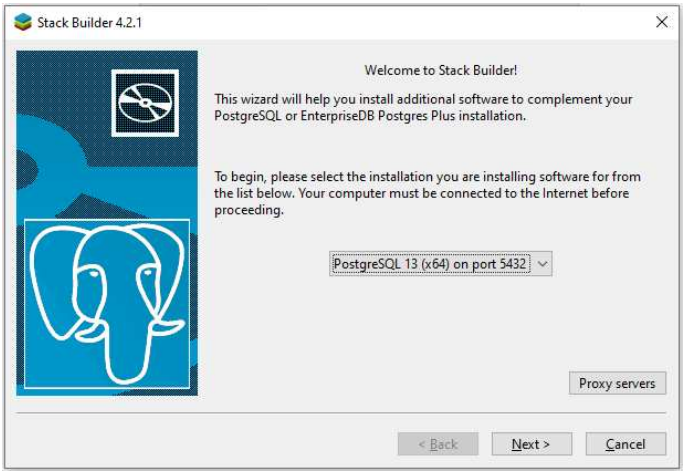
3.3 Instalacija PostGIS-a

S obzirom da su svi primjeri rađeni na Windows operacijskom sustavu, sve upute se odnose na njega.

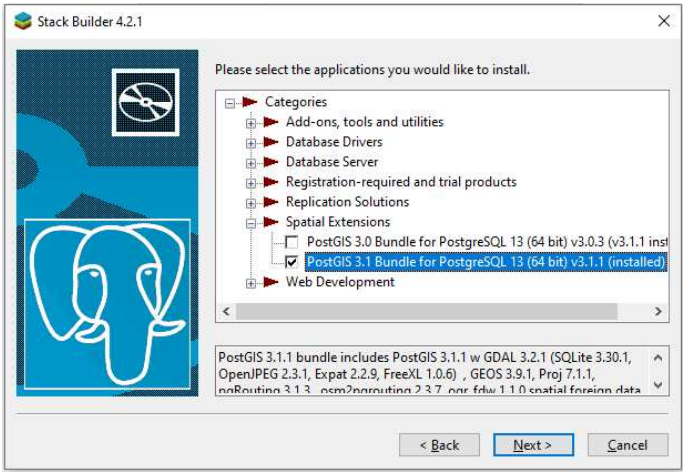
Pošto je PostGIS proširenje PostgreSQL-a, potrebno je najprije instalirati PostgreSQL. PostgreSQL se može preuzeti na sljedećem linku: `https://www.enterprisedb.com/downloads/postgres-postgresql-downloads` (odabire se određena verzija, u ovom slučaju je bila odabrana verzija 13.3.). Sama instalacija je jednostavna, potrebno je odabrati putanju za direktorij instalacije, komponente koje će biti instalirane (preporuka je označiti sve ponuđene), direktorij za pohranu podataka, unijeti zaporku za administracijskog korisnika, odabrati port te lokalne postavke. Nakon toga se pokreće aplikacija koja instalira sve potrebno na računalo. Završetkom instalacije, na računalu se nalaze: *PostgreSQL Server, pgAdmin 4, Stack Builder* i *Command Line Tools*.

Sada kada postoji PostgreSQL na računalu, može se instalirati njegovo proširenje, PostGIS. Najprije je potrebno otvoriti instalirani *Stack Builder*. Prvi korak je odabir istalirane verzije PostgreSQL-a (slika 3.1a) te potom odabir proširenja za PostGIS (slika 3.1b, odabire se aktualna verzija, u ovom slučaju 3.1.1). Nakon toga potrebno je pratiti korake,

odabrati *yes* na izbornicima gdje je to traženo i završiti instalaciju. Na taj način instalirano je PostGIS proširenje.



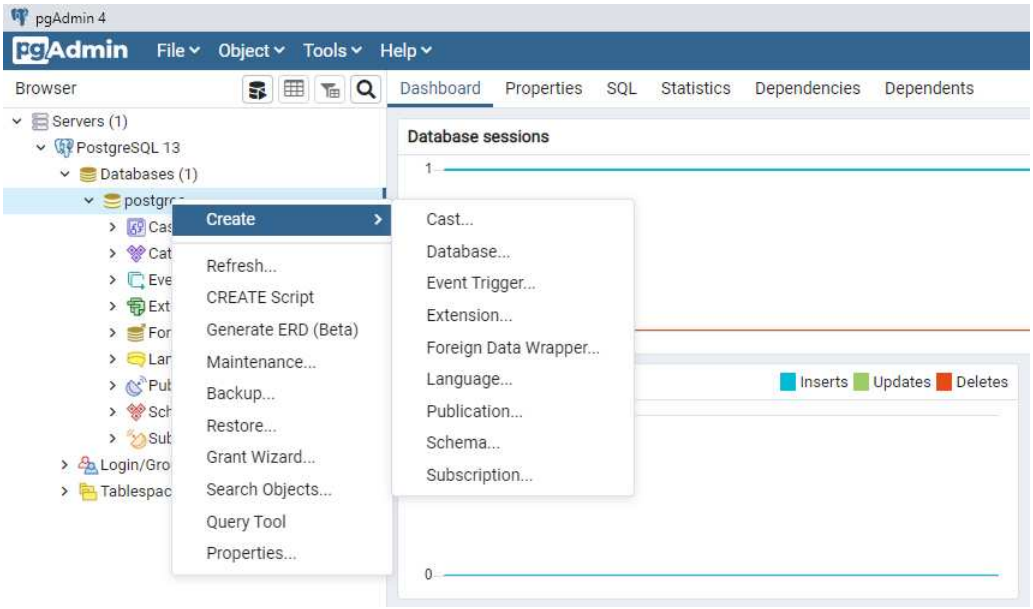
(a) Odabir instalirane verzije



(b) Odabit proširenja za PostGIS

Slika 3.1: Instalacija PostGIS proširenja

Rad s bazama podataka moguće je obavljati putem komandne linije ili instalacijom proširenja za prostorne baze podatka. S obzirom da je grafičko sučelje *pgAdmin* već instalirano (odabrano je prilikom instalacija PostgreSQL-a), njega ćemo koristiti u nastavku rada.
Pri prvom pokretanju, potrebno je upisati lozinku postavljenu prilikom instalacije PostgreSQL-a. Klikom na PostgreSQL 13 dolazi do spajanja na server te se mogu vidjeti baze podataka na tom serveru (slike 3.2).



Slika 3.2: Dodavanje PostGIS proširenja bazi

Na početku postoji samo jedna baza podataka imena postgres, neophodna je za funkcioniranje sustava i nastaje automatski instalacijom. Sada je potrebno bazi dodati PostGIS proširenje, što se može napraviti na dva načina. Prvi način prikazan je na slici 3.2. Desnim klikom na konkretnu bazu otvara se izbornik, odabire *Create* pa potom *Extension...*. U prozoru koji se otvori odabire se ekstenzija koja se želi dodati (u našem slučaju *postgis*) te klikne *Save*.
Drugi način je da se u *Query Editor* unese sljedeća naredba:

```
CREATE EXTENSION postgis;
```

Time je dodano proširenje, što se može provjeriti naredbom:

```
SELECT postgis_full_version();
```

Taj upit ispisuje verziju PostGIS-a koja je instalirana na računalo (u ovom slučaju PostGIS 3.1.1). Sada kada je sve instalirano, može se kreirati baza podataka i izvoditi upiti na njoj. Detaljni opisi kreiranja baze podataka i upita na njoj biti će prikazani pomoću primjera u poglavlju 4.

3.4 Prostorni podaci u PostGIS-u

U PostGIS-u postoje 4 vrste prostornih podataka: geometrijski, geografski, rasterski i topološki podaci. Geometrijski podaci su sadržani u PostGIS-u od samog početka, dok su ostali uvedeni kasnije. Sada ćemo ukratko opisati sve vrste podataka, a o nekima ćemo kasnije detaljnije.

- **Geometrijski podaci** - predstavlja ravninski tip podataka, prvi model za prikazivanje podataka i još uvijek najpopularniji u PostGIS-u, koristi klasični Kartezijev koordinatni sustav
- **Geografski podaci** - sferoidni geodetski tip podataka, podaci su prikazani na zakrivljenoj površini Zemlje pa su tako linije zakrivljene, a ne ravne
- **Rasterski podaci** - modeliraju prostor kao mrežu pravokutnih ćelija, svaka od njih sadrži numerički niz vrijednosti
- **Topološki podaci** - relacijski model podataka, podaci se modeliraju pomoću čvorova, granica i oblika

Geometrijski podaci

Kao što je već navedeno, PostGIS podržava geometrijske podatke od samih početaka te su oni bili jedini dostupni. Nazvani su tako jer je njihova osnova analitička geometrija. Svi podtipovi podrazumijevaju da se nalaze u Kartezijevom koordinatnom sustavu pa se tako smatra da se paralelni pravci nikad ne dodiruju, primjenjuje se Pitagorin poučak, udaljenosti koordinata su ujednačene i slično. Često se koriste geografska dužina i širina za prikazivanje geometrijskih točaka, ali to ne znači da se ne koristi Kartezijev koordinatni sustav. To bi zapravo značilo da je područje koje se promatra dovoljno malo pa se zakrivljenost Zemlje ne treba uzimati u obzir i podaci se mogu prikazati pomoću Kartezijevih koordinata. U slučaju da se promatra veće područje, potrebno je koristiti geografske podatke.

Geometrijski podaci koje promatramo su:

1. *Points* - točke
2. *Linestrings* - linije
3. *Polygons* - poligoni
4. *MultiPoints* - kolekcija točaka
5. *MultiLinestrings* - kolekcija linija

6. *MultiPolygons* - kolekcija poligona
7. *Geometry Collection* - kolekcija bilo kojih prethodno navedenih podataka

Sada ćemo objasniti svaki tip podataka te kako se oni unose u bazu i koriste.

Points

Točka se koristi za prikazivanje jedne lokacije na Zemlji. Prikazuje se koordinatama koje mogu biti u jednoj ili više dimenzija. Koriste se za prikazivanje predmeta za koje nisu bitni detalji poput oblika ili veličine. Najčešći primjer korištenja točaka je za prikazivanje gradova na karti svijeta. Točke se standardno prikazuju pomoću koordinata, `X` i `Y` za dvodimenzionalne točke te još i `Z` za trodimenzionalne. Uz to, postoji još `M` koordinata u PostGIS bazama podataka. `M` koordinata predstavlja dodatnu vrijednost koja se sprema za svaku točku. Uvijek je tipa double te se koristi za dodatne informacije potrebne za te podatke.

Tipovi točaka koji postoje za geometrijske podatke su sljedeći:

- `POINT` - točka u dvodimenzionalnom prostoru definirana `X` i `Y` koordinatama
- `POINTZ` - točka u trodimenzionalnom prostoru definirana `X`, `Y` i `Z` koordinatama
- `POINTM` - točka u dvodimenzionalnom prostoru definirana `X` i `Y` koorinatama te vrijednosti `M`
- `POINTZM` - točka u dvodimenzionalnom prostoru definirana `X`, `Y` i `Z` kooridinatama te vrijednosti `M`

U sljedećem upitu možemo vidjeti kako se kreira tablica koja sadrži razne tipove točaka te kako dodati podatke u nju.

```
CREATE TABLE points (
        id serial PRIMARY KEY,
        p geometry(POINT),
        pz geometry(POINTZ),
        pm geometry(POINTM),
        pzm geometry(POINTZM)
);

INSERT INTO points (p, pz, pm, pzm)
VALUES (
     ST_GeomFromText('POINT(1 -1)'),
```

```
     ST_GeomFromText('POINT Z(1 -1 1)'),
     ST_GeomFromText('POINT M(1 -1 1)'),
     ST_GeomFromText('POINT ZM(1 -1 1 1)')
);
```

Vidimo da je za definiranje tipa podatka korištena ključna riječ `geometry` te unutar zagrada jedan od tipova točaka koje smo prethodno definirali. Za dodavanje točaka, koristi se funkcija `ST_GeomFromText` koja iz danog teksta (dobro definiranog) vraća tip podatka `geometry`. Možemo primjetiti da je prilikom dodavanja točaka riječ `POINT` odvojena razmakom od ostatka. Ispravno je pisati i bez razmaka, ali se razmak dodaje kako bi bilo kompatibilno s drugim prostornim bazama podataka. Prilikom kreiranja stupca u bazi, u ključnoj riječi za tip podatka geometry se ne smije pojavljivati razmak, primjerice `POINTZ` se ne može zapisati kao `POINT Z`. Osim toga, trodimenzionalne vrste točaka mogu se kreirati pomoću tipa `POINT` stavljajući tri koordinate, npr. `POINT(1 1 1)`.

Linestrings

Linije čine povezane ravne crte između dvije ili više različitih točaka. Crta između točaka naziva se segment. Segmenti nisu podtipovi podataka u PostGIS-u ali moguće je da linija ima samo jedan segment. Kao i za točke, tako i za linije imamo više tipova u PostGIS-u:

- `LINESTRING` - dvodimenzionalna linija definirana s dvije ili više različitih točaka (`POINT`)
- `LINESTRINGZ` - trodimenzionalna linija definirana s dvije ili više različitih točaka (`POINTZ`)
- `LINESTRINGM` - dvodimenzionalna linija definirana s dvije ili više različitih točaka (`POINTM`)
- `LINESTRINGZM` - trodimenzionalna linija definirana s dvije ili više različitih točaka (`POINTZM`)

Linije možemo dodati na sljedeći način:

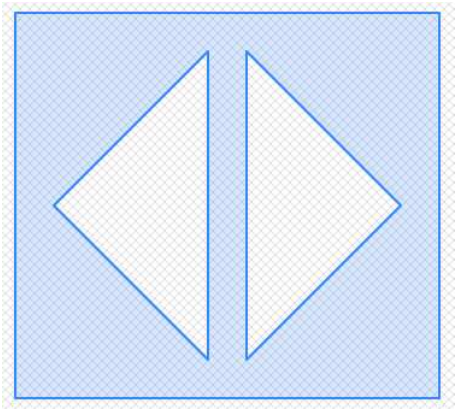
```
CREATE TABLE linestrings (
     id serial PRIMARY KEY,
     ls geometry(LINESTRING),
     lsz geometry(LINESTRINGZ),
     lsm geometry(LINESTRINGM),
     lszm geometry(LINESTRINGZM)
);
```

```
INSERT INTO linestrings (ls, lsz, lsm, lszm)
VALUES (
     ST_GeomFromText('LINESTRING(0 0, 1 1, 1 -1)'),
     ST_GeomFromText('LINESTRING Z(0 0 0, 1 1 1, 1 1 -1)'),
     ST_GeomFromText('LINESTRING M(0 0 5, 1 1 3, 1 -1 3)'),
     ST_GeomFromText('LINESTRING ZM(0 0 1 4, 0 1 1 12, 0 1 -1 7)')
);
```

Za prvu liniju koju unosimo definiramo dva segmenta. S obzirom da početna i završna točka linije nisu jednake, takvu liniju zovemo otvorenom. Kada definiramo na sljedeći način: `ST_GeomFromText('LINESTRING(0 0, 1 1, 1 -1, 0 0)')`, dobijemo liniju sa istom početnom i završnom točkom koja se zove zatvorena linija.

Polygons

Ono što čini poligon su prije spomenute zatvorene linije. Ako gledamo na primjeru trokuta, njega grade tri nekolinearne točke te bi poligon bilo područje zatvoreno linijom određenom tim trima točkama. Takva linija i sve točke koje ona zatvara čine poligon. Vanjski dio poligona nazivamo prstenom poligona. Poligon može imati najviše jedan vanjski prsten i nula ili više unutarnjih. Primjer poligona s jednim vanjskim prstenom i dva unutarnja nalazi se na slici 3.3.



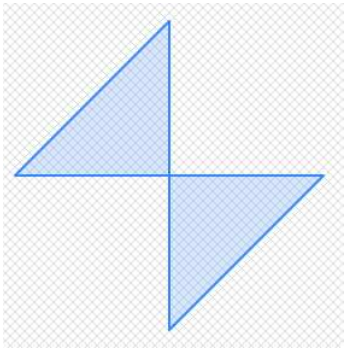
Slika 3.3: Primjer poligona s više prstena

Ovakav poligon je u bazu unesen pomoću sljedećeg upita u kojem se najprije kreira tablica te potom dodaje poligon:

```
CREATE TABLE polygons (
    id serial PRIMARY KEY,
    polygon geometry(POLYGON)
);

INSERT INTO polygons (polygon)
VALUES (ST_GeomFromText('POLYGON(
    (-0.25 -1.25,-0.25 1.25,2.5 1.25,2.5 -1.25,-0.25 -1.25),
    (2.25 0,1.25 1,1.25 -1,2.25 0),(1 -1,1 1,0 0,1 -1))')
);
```

Možemo primjetiti da je za ovakav poligon bilo potrebno definirati vanjski prsten i oba unutarnja. Prilikom inserta, najprije se definira vanjski prsten te potom unutarnji. Svaki prsten mora se pisati u zagradama kako bi se znalo koje točke pripadaju kojem prstenu. Ukoliko se definira samo vanjski prsten, nije potrebno pisati zagrade, ali se to preporuča svakako raditi. Kako bi poligon bio valjan, njegovi unutarnji prsteni se ne mogu preklapati niti mogu dijeliti granicu. Također, poligon nije u redu zadan ako jedan od unutarnjih prstena leži izvan vanjskog prstena ili ga presjeca. Takve poligone moguće je unijeti u bazu, ali postoji mogućnost da se prilikom prikazivanja istih ne može vidjeti je li taj poligon valjan ili ne. Primjerice, poligon zadan na sljedeći način: `POLYGON((2 0,0 0,1 1,1 -1, 2 0))')` je neispravan. Njegov prikaz je na slici 3.4, ali se pogledom na sliku ne može primjetiti je li valjan ili ne (mogla bi biti dva validna poligona).



Slika 3.4: Primjer neispravnog poligona

Nakon što smo definirali jednostavne vrste geografskih podataka, definirat ćemo i kolekcije podataka. Primjer korištenja kolekcija bio bi u kreiranju dražava SAD-a. Kada ne bi imali kolekcije, svaka država bi se prikazivala pomoću poligona što bi značilo 50 redaka u tablici. U svakom retku bi bio pohranjen poligon koji predstavlja tu državu. Osim toga,

za svaku državu bi trebalo dodatno definirati ostale elemente što bi zakompliciralo spremanje. Zbog toga postoje kolekcije podataka. U ovom slučaju bi svaka od država mogla postati kolekcija podataka i to multipoligon. Sada ćemo prikazati kolekcije podataka koje postoje.

Multipoints

Kolekcija točaka ne predstavlja ništa drugo nego više točaka definiranih zajedno. Takve točke definiraju se pomoću naredbe: `MULTIPOINT(0 0, 5 6, 3 -2, -3 -5)`. Zarezom se odvajaju dvije različite točke. Osim što se točke mogu odvojiti zarezom, moguće ih je staviti u zagrade, npr. `MULTIPOINT((0 0), (5 6), (3 -2), (-3 -5))`.
Kao i za točke, kolekcije točaka se mogu kreirati za više dimenzija te mogu imati `M` koordinatu koju smo definirali kod definiranja točaka. Zbog toga možemo kolekciju točaka definirati i sa sljedećim funkcijama: `MULTIPOINT Z`, `MULTIPOINT M` i `MULTIPOINT ZM`.

Multilinestrings

Kao što je *multipoint* kolekcija točaka, tako je i *multilinestring* kolekcija linija. Kolekcije linija možemo zadati na ove načine:

```
MULTILINESTRING((0 0,0 1,1 1), (-1 1,-1 -1))
MULTILINESTRING ZM ((0 0 1 1,0 1 1 2,1 1 1 3), (-1 1 1 1,-1 -1 1 2))
MULTILINESTRING M((0 0 1,0 1 2,1 1 3), (-1 1 1,-1 -1 2))
```

U ovom slučaju bi kolekcije linija definirane s `MULTILINESTRING` i `MULTILINESTRING M` izgledale isto u koordinatnom sustavu jer se koordinata `M` ne može prikazati u koordinatnom sustavu.

Multipolygons

Analogno kolekciji točaka i linija, *multipolygon* predstavlja skup više poligona. Vidjeli smo da za kolekcije linija treba paziti prilikom definiranja i postavljanja zagrada. Isto vrijedi i za kolekcije poligona. S obzirom da prilikom kreiranja poligona prva zagrada predstavlja vanjski prsten, a ostale unutarnje prstene, potrebno je i prilikom definiranja multipoligona koristiti zagrade na taj način. Bez obzira što možda neće biti unutarnjih prstena, potrebno je postaviti sve zagrade radi konzistentnosti. Primjeri definiranja kolekcije poligona su sljedeći:

```
MULTIPOLYGON(
    ((2.25 0,1.25 1,1.25 -1,2.25 0)),
    ((1 -1,1 1,0 0,1 -1))
```

```
)
MULTIPOLYGON Z(
    ((2.25 0 1,1.25 1 1,1.25 -1 1,2.25 0 1)),
    ((1 -1 2,1 1 2,0 0 2,1 -1 2))
)
MULTIPOLYGON ZM(
    ((2.25 0 1 1,1.25 1 1 2,1.25 -1 1 1,2.25 0 1 1)),
    ((1 -1 2 1,1 1 2 2,0 0 2 3,1 -1 1 4))
)
MULTIPOLYGON M(
    ((2.25 0 1,1.25 1 2,1.25 -1 1,2.25 0 1)),
    ((1 -1 1,1 1 2,0 0 3,1 -1 4))
)
```

Kao za poligone, tako i za multipoligone postoje pravila koja se moraju poštivati da bi kolekcija bila valjana. To bi bila sljedeća pravila:

- Svaki poligon u kolekciji mora biti validan sam za sebe.
- Poligoni koji čine kolekciju ne mogu se preklapati.

Geometrycollection

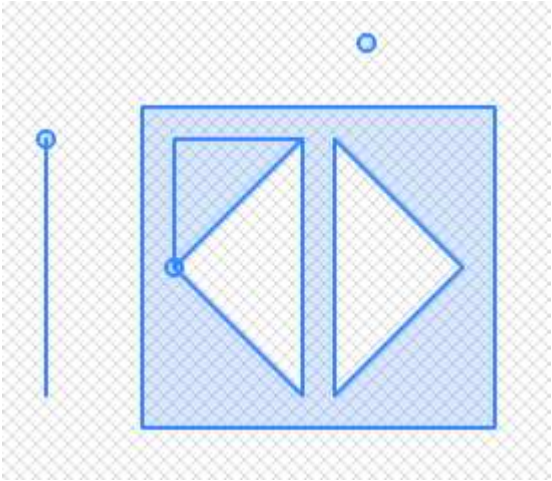
U prethodno navedenim kolekcijama vidjeli smo da elementi kolekcije moraju biti iste vrste. Cilj je spajati različite elemente te nam to omogućuje *geometrycollection*. *Geometrycollection* je podtip PostGIS-ovog tipa podataka *geometry*. Može sadržavati različite tipove podataka, bilo to točke, linije i poligoni ili njihove kolekcije. Zapravo, bilo koja vrsta geometrijskih tipova može se spremiti pomoću tipa *geometrycollection*. Sljedeće naredve predstavljaju kreiranje tablice koja sadrži takvu kolekciju te unos podataka u nju.

```
CREATE TABLE collections (
id serial PRIMARY KEY,
collection geometry(GEOMETRYCOLLECTION)
);


INSERT INTO collections (collection)
VALUES (ST_GeomFromText('GEOMETRYCOLLECTION(
    MULTIPOINT(-1 1, 0 0, 1.5 1.75),
    MULTILINESTRING((0 0, 0 1, 1 1), (-1 1, -1 -1)),
    POLYGON(
```

```
        (-0.25 -1.25, -0.25 1.25, 2.5 1.25, 2.5 -1.25, -0.25 -1.25),
        (2.25 0,1.25 1,1.25 -1,2.25 0),
        (1 -1, 1 1, 0 0, 1 -1)
        )
    )')
);
```

Tako zadana kolekcija prikazana je u koordinatnom sustavu kao na slici 3.5.



Slika 3.5: Primjer kolekcije različitih elemenata

Prilikom kreiranja baze podataka, rijetko se koriste stupci definirani kao *geometrycollection*. Razlog toga je što se većina funkcija ne može izvršavati kad kolekcijama. Na primjer, ako želimo saznati površinu poligona, to neće biti moguće ako se poligon nalazi u kolekciji s točkama i linijama. Zbog toga se *geometrycollection* najčešće koristi za prikazivanje rezultata upita (npr. traženje objekata koji se nalaze na nekom području rezultirat će kolekcijom linija, točaka i poligona).

Podaci koje smo naveli su prikazani samo pomoću ravnih linija. Osim takvih podataka, postoje i krivulje. Krivulje su se pojavile u OGC SQL/MM Part 3 specifikacijama ([11]) te je PostGIS u verziji 2.0 definirao sve ono što je propisano specifikacijama. Međutim, krivulje u PostGIS-u još uvijek nisu dovoljno razvijene te nisu u širokoj upotrebi. Iako objekti u stvarnom svijetu često imaju oblik neke krivulje, njihovo prikazivanje u bazi većinom nije pomoću krivulja. Nedostaci takog prikaza su sljedeći:

- Napredna prostorna biblioteka GEOS koju PostGIS koristi za većinu svojih funkcionalnosti ne podržava krivulje. Takav problem može se riješiti prebacivanjem kri-

vulje u linije te nakon što se obrade podaci kao linije, zapisati ih kao krivulje ako je potrebno. Međutim, takav pristup nije dovoljno točan i usporava rad.

- Neke funkcije ne podržavaju krivulje pa se za takve funkcije ponovno mora raditi pretvorba kao što je prije navedeno.
- Krivulje nisu prisutne u PostGIS-u toliko dugo kao linije pa je veća vjerojatnost da će se pojaviti neke greške tijekom rada s njima.

Bez obzira na navedene nedostatke, ovakav pristup podacima ima i svojih prednosti:

- Prikazivanje elemenata zakrivljenih u stvarnosti pomoću krivulja u bazi zauzima puno manje memorije od prikazivanja linijama. Primjerice, za prikazati krug pomoću krivulje dovoljno je zadati središte i radijus, dok bi se za istu stvar zadanu pomoću linija trebalo zadati beskonačan broj točaka.
- Sve više alata implementira podršku za krivulje pa će u budućnosti njihovo korištenje biti puno lakše.

Vrste krivulja koje su podržane u PostGIS-u su sljedeće:

1. *Circularstrings*
   Ovaj tip krivulja nastaje pomoću jednog ili više lukova za koje vrijedi da je krajnja točka jednog luka početna točka drugog luka. Lukovi se zadaju pomoću početne i završne točke te točke infleksije (npr. `CIRCULARSTRING(0 0, 2 0, 2 2, 0 2, 0 0)`). Točke s neparnim indeksima predstavljaju početne, odnosno završne točke, dok one s parnim indeksima predstavljaju točke infleksije.
   Primjer unosa krivulje u bazu:

```
INSERT INTO curves
VALUES(
 ST_GeomFromText('CIRCULARSTRING(5 5, 6 6, 4 8,
                 7 9, 9.5 9.5, 11 12, 12 12)'), null, null
);
```

2. *Compoundcurves*
   Ovakve krivulje definiraju se pomoću *circularstring*-ova i linija. Kao i za *circuarstring*-ove, završna točka jednog dijela predstavlja početnu točku idućeg dijela krivulje. U ovom slučaju, linije ne zahtjevaju riječ `LINESTRING` prilikom definiranja, definira se bez nje te se ne podržava definiranje pomoću te riječi.
   Primjer unosa krivulje u bazu:

```
INSERT INTO curves
VALUES (null,
    ST_GeomFromText(
    'COMPOUNDCURVE(
        (2 2, 2.5 2.5),
        CIRCULARSTRING(2.5 2.5, 4.5 2.5, 3.5 3.5),
        (3.5 3.5, 2.5 4.5, 3 5))'
    ), null
);
```

3. *Curvepolygons*
   Curvepolygon predstavlja poligon koji ima vanjski ili unutarnji prsten zakrivljen. Primjer unosa u bazu:

```
INSERT INTO curves
VALUES
(
 ST_GeomFromText('CURVEPOLYGON(
 (-0.5 7, -1 5, 3.5 5.25, -0.5 7),
 CIRCULARSTRING(
 0.25 5.5, -0.25 6.5, -0.5 5.75, 0 5.75, 0.25 5.5)
)'), null, null);
```

Tablica u koju su spremljene krivulje definirana je pomoću sljedeće naredbe:

```
CREATE TABLE curves (
circularString geometry(CIRCULARSTRING),
compoundCurve geometry(COMPOUNDCURVE),
curvePolygon geometry(CURVEPOLYGON)
);
```

Za sve navedene prostorne podatke, moguće je definirati i SRID broj. SRID (*Spatial Reference Identifier*) je cijeli broj koji služi za raspoznavanje različitih projekcija koordinatnog sustava. Odnosi se na primarni ključ tablice `spatial_ref_sys` u kojoj se nalazi preko 3000 poznatih prostornih referentnih sustava koji sadrže informacije potrebne za transformaciju podataka. Primjer kada je potrebno imati unesen SRID broj je kada se računaju udaljenosti između objekata. U slučaju kada je SRID jednak nuli, tada se računa euklidska udaljenost. Kada je SRID postavljen na 4326, tada se u obzir uzima zakrivljenost Zemlje te se dobije točna udaljenost između npr. dva grada.

3.5 Funkcije u PostGIS-u

Kao što je navedeno u prethodnim poglavljima, PostGIS je proširenje PostgreSQL-a koji sadrži SQL funkcije. Zbog toga je stvaranje tablice (`CREATE TABLE`), izmjena postojećeg ili dodavanje novog stupca (`ALTER TABLE`), umetanje redaka (`INSERT`), dohvaćanje podataka iz tablice (`SELECT`) i slično, isto kao i u SQL-u. Također, definirani su tipovi podataka u PostGIS-u, njihovo stvaranje te dodavanje u bazu. Sada ćemo definirati funkcije koje se koriste za rad s takvim podacima.

Jedna od klasifikacija za takve funkcije je sljedeća (prema [10]):

- Izlazne funkcije
  Pomoću ovih funkcija se prostorni objekti prikazuju u različitim formatima.
- Konstruktori
  Ove funkcije stvaraju PostGIS prostorne objekte. Mogu ih stvarati iz teksta kao što je prikazano u prethodnom poglavlju ili iz nekog drugog formata.
- Funkcije za pristup i postavljanje
  Ovakve funkcije se koriste za dohvaćanje objekta te vraćanje njegovih atributa ili postavljanje novih.
- Funkcije mjerenja
  Služe za određivanje raznih mjera između objekata, primjerice površina objekta ili udaljenost između dva objekta.
- Funkcije dekompozicije
  Ovakve funkcije iz ulaznih prostornih objekata stvaraju nove prostorne objekte tako da rastavlja zadani objekt na dijelove.
- Funkcije kompozicije
  Pomoću ovih funkcija stvaraju se novi prostornih objekti spajanjem ili grupiranjem postojećih.
- Funkcije pojednostavljenja
  Ponekad nisu potrebni svi detalji prostornih objekata pa se pomoću ovakvih funkcija dobiju samo pojednostavljeni podaci (primjerice zaokruživanje koordinata).

Izlazne funkcije

Uz pomoć ovih funkcija mogu se pregledavati objekti zapisani u standardnim formatima. Izlazne funkcije vraćaju prostorne objekte u nekom drugom standardnom formatu. Najpopularnijji izlazni formati su sljedeći: *Well-known text* - WKT, *Well-known binary* - WKB,

Geographic Mark-up LANGUAGE - GML, *Keyhole Mark-up Language* - KML, *GeoJSON*, *Scalable Vector Graphics* - SVG. Izlazne funkcije koje se koriste su sljedeće:

- `ST_AsText` ili `ST_AsEWKT` - vraćaju prikaz geometrijskih objekata u tekstualnom (WKT) formatu
- `STAsBinary` ili `ST_AsEWKB` - prikaz objekata u binarnom (WKB) formatu
- `ST_AsKML` - prikaz u KML formatu
- `ST_AsGML` - prikaz u GML formatu
- `ST_AsGeoJSON` - prikaz u GeoJSON formatu
- `ST_AsSVG` - prikaz u SVG formatu

Primjer korištenja ovih funkcija možemo vidjeti u sljedećem upitu:

```
SELECT
    ST_AsText(geom, 5) as WKT,
    ST_AsGML(geom,5) as GML,
    ST_AsKML(geom,5) as KML,
    ST_AsGeoJSON(geom,5) as GeoJSON,
    ST_AsSVG(geom,0,5) as SVG
FROM
    (SELECT
        ST_GeomFromText('LINESTRING(2 48 1,0 51 1)',4326) as geom
    ) X;
```

te se dobiju sljedeći prikazi:

- WKT: `LINESTRING Z (2 48 1,0 51 1)`
- GML:

```
<gml:LineString srsName="EPSG:4326">
        <gml:coordinates>2,48,1 0,51,1</gml:coordinates>
</gml:LineString>
```

- KML:
`<LineString><coordinates>2,48,1 0,51,1</coordinates></LineString>`

- GeoJSON:
  `"type":"LineString", "coordinates":[ [ 2, 48, 1 ], [ 0, 51, 1 ] ]`
- SVG:
  `M 2 -48 L 0 -51`

Konstruktori

Prilikom kreiranja prostornih objekata, koriste se konstruktori. Kao što se geometrijski podaci mogu ispisati u standardnom formatu, tako se i iz standardnih formata mogu dobiti geometrijski objekti. Prilikom definiranja prostornih podataka koristila se funkcija `ST_GeomFromText` koja je iz tekstualnom (WKT) formata kreirala geometrijski objekt. Sada ćemo navesti funkcije pomoću kojih se i iz ostalih formata definiraju prostorni objekti.

- `ST_GeomFromGeoJSON` - kreiranje iz GeoJSON formata
- `ST_GeomFromGML` - kreiranje iz GML formata
- `ST_GeomFromKML` - kreiranje iz KML formata
- `ST_GeomFromSVG` - kreiranje iz SVG formata
- `ST_GeomFromWKB` - kreiranje iz binarnog formata

Kao argumente, navedene funkcije primaju podatke u onom formatu iz kojeg rade prostorne objekte.

Funkcije za pristup i postavljanje

Pomoću ovakvih funkcija dolazi se do elemenata koji čine objekte. Također, postoje i funkcije koje postavljaju takve atribute. U nastavku se nalaze funkcije koje to rade:

- `ST_Centroid(geometry geom)` - vraća X i Y koordinatu centroida zadanog objekta
- `ST_CoordDim(geometry geom)` - vraća dimenziju u kojoj se objekt trenutno nalazi
- `ST_Dimension(geometry geom)` - vraća najmanju dimenziju koja je potrebna da bi cijeli objekt bio sadržan u njoj
- `ST_EndPoint(geometry geom)` - vraća krajnju točku linije, ako nije unesena linija vraća NULL
- `ST_GeometryType(geometry geom)` - vraća geometrijski tip objekta
- `ST_IsValid(geometry geom)` - provjerava valjanost objekta

- `ST_IsValidReason(geometry geom)` - ukoliko objekt nije validan, vraća kratak opis zašto nije valjan
- `ST_NPoints(geometry geom)` - vraća broj točaka koje definiraju objekt
- `ST_NumPoints(geometry geom)` - vraća broj točaka koje definiraju liniju
- `ST_SetSRID(geometry geom, integer SRID)` - postavlja SRID vrijednost objekta
- `ST_SRID(geometry geom)` - dohvaća SRID vrijednost objekta
- `ST_StartPoint(geometry geom)` - vraća početnu točku linije, ako nije unesena linije vraća NULL
- `ST_X(point P)` - vraća X koordinatu točke (analogno za Y i Z koordinatu)

Funkcije mjerenja

Kako bi odredili neka svojstva objekata ili odnose izeđu objekata, koriste se sljedeće funkcije:

- `ST_Area(geometry geom)` - vraća površinu zadanog objekta ako je poligon
- `ST_Contains(geometry geom1, geometry geom2)` - provjerava je li se sve točke objekta geom2 leže unutar objekta geom1, ukoliko je to istina, vraća `TRUE`
- `ST_Disjoint(geometry geom1, geometry geom2)` - za zadane objekte provjerava je li njihov presjek prazan i ako je, vraća `TRUE`
- `ST_Distance(geometry geom1, geometry geom2)` - vraća najmanju udaljenost između zadanih objekata
- `ST_Equals(geometry geom1, geometry geom2)` - uspoređuje zadane objekte te vraća `TRUE` ako su istog tipa i imaju iste koordinate
- `ST_Intersects(geometry geom1, geometry geom2)` - uspoređuje zadane objekte te vraća `TRUE` ako se sijeku
- `ST_Length(geometry geom)` - vraća duljinu zadanog objekta ako je linija, kolekcija linija, krivulja ili kolekcija krivulja
- `ST_Overlaps(geometry geom1, geometry geom2)` - vraća `TRUE` ako se zadani objekti sijeku ali nisu potpuno sadržani jedan u drugom

- `ST_Perimeter(geometry geom)` - vraća opseg zadanog objekta ako je poligon ili kolekcija poligona

Navedene funkcije rade s objektima u 2D prostoru. Ekvivalentne funkcije za objekte u 3D prostoru su sljedeće: `ST_3DLength, ST_3DPerimeter` i `ST_3DArea`.

Funkcije dekompozicije

Često je potrebno za pojedine objekte izdvojiti neke njihove djelove. Za to služe funkcije dekompozicije. Navodimo one koje se najčešće koriste:

- `ST_Boundary(geometry geom)` - vraća granice objekta, za linije su to točke, dok su za poligone to linije
- `ST_Difference(geometry geom1, geometry geom2)` - vraća objekt koji je razlika geom1 i geom2 (sadrži dio objekta geom1 koji se ne siječe s objektom geom2)
- `ST_ExteriorRing(geometry geom)` - vraća vanjski prsten poligona
- `ST_GeomertyN(geometry geom, integer n)` - vraća n-ti objekt zadane kolekcije objekata
- `ST_InteriorRingN(geometry geom, integer n)` - vraća n-ti unutarnji prsten poligona (indeksi kreću od 1)
- `ST_Intersection(geometry geom1, geometry geom2)` - vraća presjek dvaju zadanih objekata
- `ST_NRings(geometry geom)` - vraća broj prstenova za poligone ili kolekcije poligona
- `ST_NumGeometries(geometry geom)` - vraća broj objekata u kolekciji
- `ST_PointN(geometry geom, integer n)` - vraća n-tu točku u liniji ili kolekciji linija, negativni brojevi predstavljaju točke unatrag pa je tada -1 zadnja točka
- `ST_SymDifference(geometry geom1, geometry geom2)` - vraća objekt koji predstavlja uniju objekata bez njihovog presjeka
- `ST_Union(geometry geom1, geometry geom2)` - iz zadanih argumenata vraća uniju, mogu biti zadana dva objekta ili lista više objekata

Funkcije kompozicije

Prilikom definiranja konstruktora objekte smo stvarali iz različitih formata. Pomoću funkcija kompozicije, objekti se stvaraju iz drugih objekata. Funkcije za takvo stvaranje objekata su sljedeće:

- `ST_MakeEnvelope(float xmin, float ymin, float xmax, float y max, integer srid)` - kreira pravokutnik s definiranom minimalnom i maksimalnom vrijednosti koordinata te SRID-om
- `ST_MakePolygon(geometry geom)` - iz zadane zatvorene kolekcije linija stvara poligon s tom linijom kao vanjskim prstenom, osim vanjskog prstena mogu biti zadani i unutarnji
- `ST_Multi(geometry geom)` - pretvara objekt u kolekciju (POINT pretvara u MULTIPOINT, LINESTRING u MULTILINESTRING, POLYGON u MULTIPOLYGON)
- `ST_Point(float x, float y)` - kreira točku iz zadanih x i y koordinata, funkcija `ST_MakePoint` kreira točku kao i `ST_Point` ali može imati zadanu z koordinatu i m koordinatu

Poglavlje 4

Stvarni primjer prostorne baze podataka

4.1 Opis baze podataka

Nakon što je definirana prostorna baza podataka i opisane njene mogućnosti na primjeru PostGIS baze podataka, sada će se prikazati stvarni primjer kreiranja i korištenja takve baze. Baza služi za unapređenje aplikacije PayDo tvrtke Infoart. Aplikacija PayDo je web i mobilna aplikacija koja se koristi za plaćanje parkinga u Hrvatskoj. Cilj baze podataka je ubrzati i olakšati pronalaženje grada, odnosno zone u kojoj se korisnik nalazi. Također, potrebno je otkriti postoje li preklapanja unutar zona te ako postoje, ispisati koje su to zone te na kojim mjestima se nalaze preklapanja.

Podaci s kojima je potrebno raditi nalaze se u bazi u tablicama `CITY` i `PARKING_ZONE`. Za unos u našu bazu podataka, potrebno je napravit export stupaca `CODE` i `BORDER_DATA` iz tablice `CITY` te `CODE`, `CITY_CODE` i `POLYGON_DATA` iz tablice `PARKING_ZONE`. `BORDER_DATA` i `PARKING_ZONQ` zapisani su u GeoJSON formatu, format baziran na JSON-u koji omogućuje spremanje jednostavnih geografskih podataka. Primjer takvih podataka za grad Ston je 'ston' kao `CODE` te kao `BORDER_DATA` sljedeći GeoJSON (prikazan je samo dio podataka jer GeoJSON sadrži preko 17000 linija):

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
```

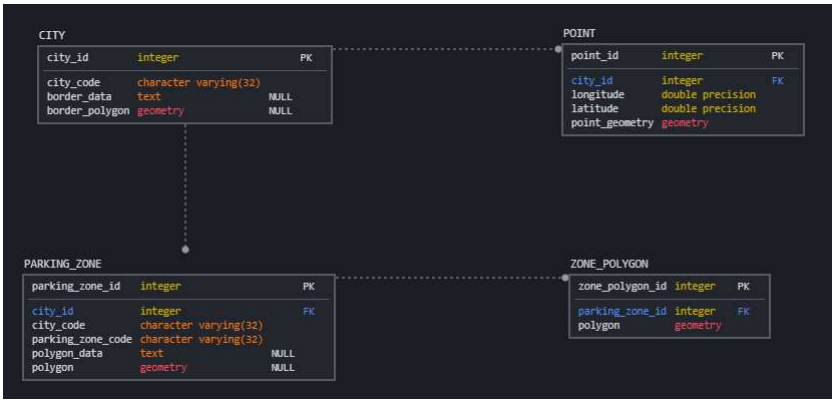
```
        "city_code": "ston",
        "name": "Ston",
        "descrpition": "City"
      },
      "geometry": {
        "type": "MultiPolygon",
        "coordinates": [
          [
            ...
          ]
        ]
      }
    }
  ]
}
```

Također, primjer jedne zone za grad Ston je sljedeći: `CODE`: 'STON_3', `CITY_CODE`: 'ston' te `POLYGON_DATA`:

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "MultiPolygon",
        "coordinates": [
          [
            [
              [
                17.696643,
                42.836042
              ],
              ...
              [
                17.696643,
                42.836042
              ]
            ]
          ]
```

```
        ]
      },
      "properties": {
        "city_code": "ston",
        "zone_code": "STON_3",
        "zone_name": "Zona 3",
        "fill": "#2FB87C",
        "fill-opacity": 0.5,
        "working_hours": ""
      }
    }
  ]
}
```


Ovi podaci spremaju se u CSV file te se iz takvog prikaza učitaju u našu bazu. Baza se sastoji od tablica, atributa i veza kako je prikazano na ER dijagramu 4.1.



Slika 4.1: ER dijagram baze podataka

Iz dijagrama vidimo da se baza sastoji od četiri tablice, `CITY`, `PARKING_ZONE`, `POINT` i `ZONE_POLYGON`. Tablica `CITY` predstavlja tablicu ekvivalentnu onoj u matičnoj bazi iz koje se koriste podaci. U nju se za svaki grad zapisuje `city_id` koji je primarni ključ, `code` dobiven iz matične tablice, `border_data` također dobiven iz matične tablice te `border_polygon`. `border_polygon` dobije se iz GeoJSON-a `border_data` tako da se dohvati dio 'geometry' te pretvori u tip PostGIS tip geometry.
Za svaki grad iz tablice `CITY` generiraju se točke u tablicu `POINT`. Točke koje se generiraju služe za brže pronalaženje grada u kojem se korisnik nalazi. Generiranje točaka radi se za svaki grad koji je unesen u tablicu na način da se zemljopisne koordinate točaka razlikuju

na treću decimalu (primjerice točke: (15.123, 45.234) i (15.123, 45.235)). Na taj način se dobije dovoljan broj točaka pomoću kojih se može odrediti grad u kojem se korisnik nalazi. Za svaku točku se sprema id grada kojem točka pripada (`city_id`), koordinate, odnosno `longitude` i `latitude` te geometrijski prikaz točke `point_geometry`. Zapis točke pomoću koordinata i geometrijskog tipa podataka služi za usporedbu brzine upita, što će biti prikazano nešto kasnije u radu.
Sljedeća tablica je `PARKING_ZONE`. Kao i za gradove, tako i ova tablica predstavlja zone iz matične tablice. U polje `parking_zone_code` zapisuje se `code` iz matične tablice, `city_code` predstavlja kod grada koji se također nalazi u matičnoj tablici te `polygon_data` predstavlja GeoJSON s podacima o konkretnoj zoni. Iz GeoJSON podatka o zonama dobije se `polygon` tipa geometry koji prikazuje zonu. Tablica još sadrži `city_id` pomoću kojeg se povezuje na tablicu `CITY`.
U tablicu `ZONE_POLYGON` spremaju se pojedinačni poligoni za zone. Pošto svaka zona može imati jedan ili više poligona, u ovu tablicu se sprema svaki poligon posebno. Za svaku zonu se `polygon` iz tablice `PARKING_ZONE` raščlanjuje na pojedninačne poligone te uz `parking_zone_code` sprema u tablicu `ZONE_POLYGON`. Ova tablica služi za određivanje točnog poligona najbližeg korisniku za parkiranje.

Sada kada imamo opis svih tablica, možemo prikazati kreiranje baze, tablica i unos podataka u tablice.

4.2 Kreiranje baze u pgAdmin-u

Kao što smo prethodno rekli, svi primjeri će biti rađeni pomoću PostGIS-a u pgAdminu. Za kreiranje baze potrebno je kliknuti desnim klikom na *Databases*. U izborniku koji se otvori treba odabrati *Create* i potom *Database...*. U novootvorenom prozoru upisuje se naziv baze, u ovom slučaju paydo i sprema se napravljena baza.

Nakon napravljene baze podataka, moguće je kreirati tablice. Njih kreiramo tako da otvorimo *Query Tool*. Kako bi mogli kreirati tablice s prostornim podacima, potrebno je dodati postgis ekstenziju za kreiranu bazu.

Sljedeći kod predstavlja kreiranje tablica.

```
1  CREATE TABLE city (
2          city_id SERIAL PRIMARY KEY,
3          city_code VARCHAR(32) UNIQUE NOT NULL,
4          border_data TEXT,
5          border_polygon GEOMETRY
6  );
7
8  CREATE TABLE point (
9          point_id SERIAL PRIMARY KEY,
10         city_id INTEGER NOT NULL,
11         longitude DOUBLE PRECISION,
```

```
12          latitude DOUBLE PRECISION,
13          point_geometry GEOMETRY,
14          CONSTRAINT fk_city_id
15                  FOREIGN KEY (city_id)
16                          REFERENCES city(city_id)
17  );
18
19  CREATE TABLE parking_zone (
20          parking_zone_id SERIAL PRIMARY KEY,
21          city_id INTEGER NOT NULL,
22          city_code VARCHAR(32) NOT NULL,
23          parking_zone_code VARCHAR(64) UNIQUE NOT NULL,
24          polygon_data TEXT,
25          polygon GEOMETRY,
26          CONSTRAINT fk_city_id
27                  FOREIGN KEY (city_id)
28                          REFERENCES city(city_id)
29  );
30
31  CREATE TABLE zone_polygon (
32          zone_polygon_id SERIAL PRIMARY KEY,
33          parking_zone_id INTEGER NOT NULL,
34          polygon GEOMETRY,
35          CONSTRAINT fk_parking_zone_id
36                  FOREIGN KEY (parking_zone_id)
37                          REFERENCES parking_zone(parking_zone_id)
38  );
```

Ovakvim naredbama kreirane su prethodno navedene tablice. Polja `city_id`, `point_id`, `parking_zone_id` i `zone_polygon_id` redom su primarni ključevi za navedene tablice. U tablici `POINT` postoji strani ključ `city_id` koji predstavlja vezu s tablicom `CITY`, isti strani ključ postoji za tablicu `PARKING_ZONE` te za tablicu `ZONE_POLYGON` postoji strani ključ `parking_zone_id` kao veza za tablicu `PARKING_ZONE`. Primarni ključevi su `SERIAL` tipovi podataka što bi značilo da se za njih automatski generira jedinstvena vrijednost tipa integer koja nikad nije `null`.

Na ovako kreirane tablice postavljeni su triggeri (procedura koja se izvršava nakon određenog događaja nad bazom, npr. unos ili izmjena podataka). Pošto se samo dio podataka unosi iz CSV file-a, potrebno je definirati triggere koji omogućuju unos ostalih podataka prilikom unosta podataka iz CSV file-a. Najprije ćemo prikazati triggere definirane na tablici `CITY`. Za definiranje triggera prvo je potrebno definirati funkciju koja će se obavljati. Ona se definira pomoću PostgreSQL-ovog proceduralnog jezika `plpgsql` koji podržava SQL-ove funkcije, triggere, petlje te razne druge funkcije. Prva funkcija je `setBorderPolygon()` definirana na sljedeći način:

```
1  CREATE OR REPLACE FUNCTION setBorderPolygon()
2  RETURNS TRIGGER
3  AS $
4    BEGIN
5      IF new.border_polygon IS NULL THEN
6        new.border_polygon  = (
7          ST_GeomFromGeoJSON(new.border_data::JSON#>'{features, 0, geometry}')
```

```
8       );
9     END IF;
10   RETURN NEW;
11 END; $$
12 LANGUAGE 'plpgsql';
```

Pomoću te funkcije postavlja se vrijednost `border_polygon` tako da se dohvati geometry dio unesenog GeoJSON-a `border_data`. Pomoću te funkcije definira se trigger:

```
1 CREATE TRIGGER setBorderPolygonTrigger
2 BEFORE INSERT ON city
3 FOR EACH ROW
4 EXECUTE PROCEDURE setBorderPolygon();
```

Ovako definiran trigger prije nego unese podatke u bazu, za svaki redak izvrši funkciju `setBorderPolygon()` i tek nakon toga unosi podatke u bazu. Na istoj tablici potrebno je još definirati trigger koji će omogućiti unošenje točaka u tablicu `POINT`. Najprije definiramo funkciju koja generira same točke. Ona prima `border_polygon` i decimalni broj koji određuje udaljenost točaka, u našem slučaju 0.001.

```
1  CREATE OR REPLACE FUNCTION generateNewPoints(geometry, numeric)
2  RETURNS geometry AS $
3          SELECT
4                  ST_Collect(ST_SetSRID(ST_POINT(x/1000::float,y/1000::float),ST_SRID($1)))
5           FROM
6                  generate_series(floor(ST_XMin($1)*1000)::int,
7                                  ceiling(ST_XMax($1)*1000)::int,$2*1000) as x ,
8                  generate_series(floor(ST_YMin($1)*1000)::int,
9                                  ceiling(ST_YMax($1)*1000)::int,$2*1000) as y
10         WHERE
11                 ST_Intersects($1, ST_SetSRID(ST_POINT(x/1000::float,y/1000::float),
12                                 ST_SRID($1)));
13 $$
14 LANGUAGE sql;
```

Funkcija radi na sljedeći način: pomoću funkcije `generate_series` generira brojeve tako da je razlika među njima zadana udaljenost. Početna točka je najmanja X, odnosno Y koordinata, a završna je najveća X, odnosno Y koordinata. Potrebno je na početku te koordinate pomnožiti s 1000 jer `generate_series` radi s podacima tipa integer. Množi se s 1000 jer se udaljnost gleda na treću koordinatu te tako dobijemo integer ekvivalentan tome. Takve točke ”sprema” u varijable x i y te pomoću njih radi točke sa SRID vrijednošću zadane geometrije. U tom koraku potrebno je ponovno podijeliti točke s 1000 kako bi ponovno dobili geografske koordinate. Pomoću tako definirane funkcije, definira se sljedeća funkcija za trigger:

```
1 CREATE OR REPLACE FUNCTION generatePoints()
2 RETURNS TRIGGER
3     AS $
4         BEGIN
5             INSERT INTO point(city_id, longitude, latitude, pointGeometry)
6             SELECT
```

```
7                 city_id,
8                 ST_X((generatedPoint).geom),
9                 ST_Y((generatedPoint).geom),
10                (generatedPoint).geom
11            FROM (
12                SELECT
13                    new.city_id,
14                    new.city_code,
15                    ST_DumpPoints(generateNewPoints(ST_GeomFromGeoJSON(new.border_data::
                          JSON#>'{features, 0, geometry}'), 0.001)) as generatedPoint
16                ) as points;
17    RETURN NEW;
18 END; $$
19 LANGUAGE 'plpgsql';
```

Ovako definirana funkcija koristi prethodno definiranu `generateNewPoints()`. Točke koje se dobiju pomoću nje unose se u tablicu `POINT`. Kreiranje triggera koji izvršava tu funkciju radi se pomoću sljedećeg upita:

```
1 CREATE TRIGGER generatePointsTrigger
2 AFTER INSERT ON city
3 FOR EACH ROW
4 EXECUTE PROCEDURE generatePoints();
```

Za razliku od triggera `setBorderPolygonTrigger`, ovaj trigger se izvršava tek nakon unosa podataka u tablicu `CITY`. Razlog toga je što već mora biti definiran `city_id` (on je strani ključ) kako bi se redak ubacio u tablicu.

Nakon što su definirani triggeri za tablicu `CITY`, sada ćemo definirati one za tablicu `PARKING_ZONE`. Kao i za prethodnu tablicu, potrebno je definirati podatke koji se ne unose iz CSV file-a. U ovom slučaju to je `polygon` i `city_id`. `polygon` se dobije iz `polygon_data` na isti način kao i `border_polygon` u tablici `CITY`, dok se `city_id` dobije dohvaćanjem iz tablice `CITY`. Funkcija koja to sve obavlja je sljedeća:

```
1  CREATE OR REPLACE FUNCTION setZoneProperties()
2  RETURNS TRIGGER
3  AS $
4      BEGIN
5          new.polygon := ST_GeomFromGeoJSON(new.POLYGON_DATA::JSON#>'{features, 0, geometry}
               ');
6          new.city_id = (
7              SELECT city_id
8              FROM city
9              WHERE city_code LIKE new.city_code
10         );
11     RETURN NEW;
12 END; $$
13 LANGUAGE 'plpgsql';
```

Trigger se postavlja tako da se podaci postavljaju prije unosa u bazu, analogno triggeru `setBorderPolygonTrigger` za prethodnu tablicu.

Još jedan trigger definiran na ovoj tablici je onaj koji unosi podatke u tablicu `ZONE_POLYGON`. Funkcija koja to radi je sljedeća:

```
CREATE OR REPLACE FUNCTION generateZonePolygons()
RETURNS TRIGGER
AS $
    BEGIN
        INSERT INTO zone_polygon(parking_zone_id, polygon)
        SELECT
            parking_zone_id,
            poligon
        FROM (
            SELECT
                new.parking_zone_id,
                (ST_Dump(ST_MakeValid(ST_GeomFromGeoJSON(new.polygon_data::JSON#>'{
                    features, 0, geometry}')))).geom as poligon) as pol;
    RETURN NEW;
END; $$
LANGUAGE 'plpgsql';
```

Ona za sve poligone koji predstavljaju parking zone generira pojedinačne poligone te njih i id zone unosi u tablicu `ZONE_POLYGON`. Takav trigger se obavlja nakon unosa retka u tablicu `PARKING_ZONE` kako bi već postojao `parking_zone_id` koji je potreban za unos.

Tako su definirane sve potrebne tablice i moguće je unijeti podatke u njih. Prilikom unosa podataka potrebno je napraviti sljedeće:

- desnim klikom kliknuti na tablicu u koju se žele unijeti podaci
- u otvorenom izborniku odabrati *Import/Export...*
- u otvorenom prozoru najprije označiti *Import* te potom odabrati datoteku iz koje se unose podaci, mogu se odabrati i dodatne opcije koje ovise o datoteci iz koje se učitava
- u izborniku *Columns* odabrati stupce u koje se unose podaci

Sada kada imamo definirane tablice i unesene podatke, moguće je stvarati upite na toj bazi.

4.3 Upiti na bazi

Najprije na bazi želimo provjeriti valjanost svih podataka. Prvi korak u tome je provjeriti jesu li poligoni dobro definirani. Najprije provjeravamo podatke iz tablice `CITY` pomoću sljedećeg upita:

```
SELECT
    city_code,
    ST_IsValidReason(border_polygon)
FROM city
WHERE ST_IsValid(border_polygon) IS FALSE
```

Ovaj upit ispisati će sve kodove gradova za koje vrijedi da njihovi poligoni nisu dobro zadani. U našem slučaju dobijemo sljedeće podatke: `tisno, Hole lies outside shell`. To bi značilo da poligon za grad Tisno nije dobro zadan jer su unutarnji prsteni poligona zadani izvan vanjskog poligona. Kako bi taj problem riješili, moramo napraviti update za taj redak, što radimo na sljedeći način:

```
1 UPDATE city
2 SET border_polygon = ST_MakeValid(border_polygon)
3 WHERE ST_IsValid(border_polygon) IS FALSE
```

Sada kada imamo dobro definiran svaki grad pojedinačno, postrebno je provjeriti postoji li presjek između dva različita grada. Takva provjera radi se pomoću sljedećeg upita:

```
1 SELECT DISTINCT
2     city1.city_code as code1,
3     city2.city_code as code2
4 FROM CITY city1
5     JOIN CITY city2 ON city1.city_id < city2.city_id
6 WHERE ST_Intersects(city1.border_polygon, city2.border_polygon)
7 AND ST_Touches(city1.border_polygon, city2.border_polygon) IS FALSE
```

Ovaj upit radi sljedeće: uzima dva različita grada iz tablice `CITY` i ispisuje ih ako presjek njihovih poligona nije prazan te se ne dodiruju. Uvjet dodirivanja je potreban jer PostGIS za dodirivanje vraća da postoji neprazan presjek. U stvarnom svijetu, ako se poligoni dodiruju to znači da imaju zajedničku granicu pa je taj slučaj potrebno ignorirati. Pomoću ovog upita dobijemo sljedeće parove kodova gradova:

Code 1	Code 2
murter	tisno
sibenik	vodice
solin	split
trigir	okrug_gornji

U ovom slučaju ne možemo riješiti problem preklapanja pomoću funkcije ST_MakeValid jer bi se tada izgubilo područje koje je u presjeku. Zbog toga je potrebno ručno to napraviti i odrediti kojem gradu to područje pripada. Analogno se može napraviti i za zone. Potrebno je još za uvjet preklapanja dviju zona dodati da im se podudaraju `city_id`-evi.

Sada kada u bazi imamo valjane podatke, možemo raditi upite nad njima. Jedan od takvih upita je pronalaženje najbliže zone za zadanu točku (koja zapravo predstavlja poziciju korisnika). Upit koji pronalazi takvu zonu je sljedeći:

```
1 CREATE OR REPLACE FUNCTION getNearestParkingZone(lon numeric, lat numeric)
2 RETURNS TABLE(parking_zone_id integer, parking_zone_code character varying, polygon
      geometry)
3 AS $
4     DECLARE
5         cityId INTEGER := getCityId(lon, lat);
6     BEGIN
```

```
7         RETURN QUERY
8             SELECT
9                 parking_zone.parking_zone_id,
10                parking_zone.parking_zone_code,
11                ST_Collect(ST_SetSRID(ST_MakePoint(lon, lat), 4326), parking_zone.polygon)
12            FROM parking_zone
13            WHERE
14                city_id = cityId
15            ORDER BY ST_Distance(ST_SetSRID(ST_MakePoint(lon, lat), 4326)::geography,
                  parking_zone.polygon::geography);
16    END;
17 $$
18 LANGUAGE 'plpgsql';
```

Najprije je za tu funkciju definirana funkcija `getCityId(numeric, numeric)` koja za zadanu dužinu i širinu nalazi grad u kojem se ta točka nalazi. Implementacija te funkcije je sljedeća:

```
1  CREATE OR REPLACE FUNCTION getCityId(lon numeric, lat numeric)
2  RETURNS INTEGER
3  AS $
4      SELECT
5          city_id
6      FROM point
7      WHERE
8          longitude = Round(lon, 3)
9          AND latitude = Round(lat, 3)
10 $$
11 LANGUAGE sql
12 IMMUTABLE
13 RETURNS NULL ON NULL INPUT;
```

Vidimo da se dužina i širina zaokružuju na tri decimale pomoću funkcije `Round`. To se radi zbog toga što su u tablici `POINT` koordinate zaokružene na tri decimale te se na taj način može pretraživati tablica. Kada imamo ID grada, možemo tražiti najbližu zonu u tom gradu. Prikazujemo `parking_zone_id, parking_zone_code` te geografski prikaz nađene zone i unesene točke. U ovoj funkciji biti će prikazane sve zone u tom gradu soritrane po udaljenosti od točke. Udaljenost računamo pomoću funkcije `ST_Distance` koja prima zadanu točku i poligon grada. Na primjeru točke s dužinom i širinom (15.977, 45.824) dobili bi ispis podataka kao na slici 4.2a te geografski prikaz najboljeg razultata kao na slici 4.2b.

357	ZAGREB_1	0107000020E6100...
358	ZAGREB_2	0107000020E6100...
360	ZAGREB_1-1	0107000020E6100...
356	ZAGREB_2-3	0107000020E6100...
362	ZAGREB_4-2-1	0107000020E6100...
361	ZAGREB_4-2	0107000020E6100...
459	ZAGREB_2-2	0107000020E6100...
359	ZAGREB_3	0107000020E6100...
342	ZAGREB_4-1	0107000020E6100...
441	ZAGREB_1-2	0107000020E6100...
363	ZAGREB_SVE	0101000020E6100...

(a) Rezultat pretraživanja najbližih zona

(b) Prikaz najbliže zone

Slika 4.2: Rezultati funkcije `getNearestZone(15.977, 45.824)`

Osim ovako definirane funkcije, može se definirati funkcija `getNearestZoneId(numeric, numeric)` koja za zadanu dužinu i širinu vrati samo ID zone koja je najbliža zadanoj točki. Tako definirana funkcija služi kako bi se mogla definirati funkcija koja će pronalazit poligon zone koji je najbliži točki. Funkcija za dohvaćanje najbližeg poligona izgleda ovako:

```
1  CREATE OR REPLACE FUNCTION getNearestParkingPolygon(lon numeric, lat numeric)
2  RETURNS TABLE(zone_polygon_id integer, polygon geometry)
3  AS $
4      DECLARE
5          zoneId INTEGER := (SELECT parking_zone_id FROM getNearestParkingZoneId(lon, lat));
6      BEGIN
7          RETURN QUERY
8              SELECT
9                  zone_polygon.zone_polygon_id,
10                 ST_Collect(ST_SetSRID(ST_MakePoint(lon, lat), 4326), zone_polygon.polygon)
11             FROM zone_polygon
12             WHERE
13                 parking_zone_id = zoneId
14             ORDER BY ST_Distance(ST_SetSRID(ST_MakePoint(lon, lat), 4326)::geography,
                   zone_polygon.polygon::geography)
15             LIMIT 1;
16     END; $$
17 LANGUAGE 'plpgsql'
```

dok prikaz točke i najbliže zone izgleda kao na slici 4.3.

Slika 4.3: Prikaz razultata funkcije `getNearestParkingPolygon(15.977, 45.824)`

4.4 Indeksiranje podataka

Sve tablice koje su do sada bile definirane nisu imale indeksirane podatke. U ovoj odjeljku usporedit ćemo vremena izvođena funkcija na neindeksiranim podacime te s dvije vrste indeksa. Funkcija pomoću koje ćemo uspoređivat biti će ona za pronalaženje najbliže zone, `getNearestParkingZone(numeric, numeric)`. Za vrijeme izvršavanja uzimamo prosječno vrijeme za pet različitih izvršavanja. Takvo vrijeme za izvršavanje funkcije nad tablicom bez indeksa iznosi 740 milisekundi.
Najprije ćemo vidjeti postiže li se ubrzanje kada se postavi indeks na `longitude` u tablici `POINT`. S obzirom da ta tablica sadrži 1100000 redaka, možemo pretpostaviti da će ubrzanje biti najveće kada postavimo indeks na neki od njenih podataka. Indeks postavljamo na sljedeći način:

```
1 CREATE INDEX longitudeIndex
2 ON point(longitude)
```

Sada kada pokrećemo funkciju nad istim podacima i s istim ulaznim podacima pet puta, dobijemo prosječno izvršavanje funkcije 106,7 milisekundi. Vidimo da s tako postavljenim indeksom dobijemo veliko ubrzanje.

Sada ćemo vidjeti koliko se ubrzanje dobije postavljanjem indeksa na prostorne podatke. Kako bi imalo smisla to promatranje, potrebno je preoblikovati funkciju tako da pretražuje tablicu `POINT` pomoću prostornih podataka tj. točaka, a ne pomoću dužine i širine tj. numeric tipa podataka. Takvo pretraživanje podataka događa se u funkciji `getCityId` pa ćemo nju izmjeniti tako da je njena implementacija sljedeća:

```
CREATE OR REPLACE FUNCTION getCityId(lon numeric, lat numeric)
RETURNS INTEGER
AS $
    SELECT
        city_id
    FROM point
    WHERE
        ST_Equals(ST_SetSRID(ST_MakePoint(lon, lat), 4326), point_geometry)
$$
LANGUAGE sql
IMMUTABLE
RETURNS NULL ON NULL INPUT;
```

Pomoću tako definirane funkcije, prosječno izvođenje u pet pokretanja iznosi 1 sekundu i 308 milisekundi. Takvo izvršavanje upita znatno je sporije nego izvršavanje pomoću pronalaska `city_id`-a pomoću koordinata. Sada ćemo kreirati indeks na tablici, ali ovaj put na `point_geometry`. Takvo postavljanje radimo na sljedeći način:

```
CREATE INDEX "pointIndex"
ON point USING gist
(point_geometry gist_geometry_ops_nd)
TABLESPACE pg_default;
```

Ako sada gledamo vrijeme izvršavanja kao u prethodnim slučajevima, dobijemo da izvršavanje prosječno traje 106,4 milisekundi.

Iz ovoga možemo zaključiti da je izvršavanje upita bez indeksa na tablicama veoma sporo. To se najviše primjećuje kada se pretražuje po geometrijskim podacima te je ubrzanje u tom slučaju najveće. Izvršavanje funkcija s indeksima na različitim podacima se zapravo i ne razlikuje. U oba slučaja smo dobili skoro pa iste rezultate te možemo zaključiti da je u ovom slučaju svejedno koje indekse koristimo, dok to ne mora biti slučaj za sve baze podataka. Također, vidimo da izvršavanje upita na geometrijskim podacima može potrajati jako dugo ukoliko nema indeksa. Vrijeme izvršavanja ne bi se osjetilo na manjim tablicama, ali na imalo većim tablicama dolazi do osjetne razlike u vremenima izvršavanja upita.

Zaključak

Nakon što je tema prostornih baza podataka obrađena i prikazano korištenje prostornih baza podataka, možemo uočiti prednosti i nedostatke takvih baza podataka.

Kao najveću prednost prostornih baza podataka možemo navesti vizualizaciju podataka. Na jednostavan način možemo prikazati veću količinu podataka tako da ih korisnik može lakše koristiti. Vizualizacija se može napraviti na više načina tako da odgovara zahtjevima korisnika. Također, analiza takvih podataka je veoma jednostavna jer postoji mnogo funkcija koje se mogu koristiti za obradu te se podatke tretira kao bilo koje druge u bazi. Osim toga, jedna od prednosti je implementiranje vlastitih funkcija za rad s podacima.

Nedostatak ovakvih baza podataka je ponajprije cijena softvera jer zahtjeva veliku količinu podataka kako bi bio koristan za izvršavanje nekih zadataka. Samo prikupljanje i unos podataka može potrajati te tako povećati cijenu. S većom količinom podataka, raste veličina potrebnog prostora za pohranu te i sama cijena. Osim toga, može doći do krivih izračuna ako se ne uzima u obzir zakrivljenost Zemlje. U nekim slučajevima to nije problem, ali često može biti te nije lako primjetiti takvu pogrešku.

Bez obzira na navedene nedostatke, prostorne baze podataka su sve više zastupljene u današnjim tehnologijama. Primjenu možemo vidjeti prvenstveno u geografiji i kartografiji. Svako moderno prikazivanje geografskog prostora u pozadini ima prostornu bazu podataka. Osim toga, primjenu možemo vidjeti u geodeziji, prometu, turizmu i sličnim djelatnostima. Olakšavaju spremanje i korištenje velike količine podataka te tako zamjenjuju starije metode. Modernizacijom sustava će prostorne baze podataka sve popularnije te bi njihova upotreba u praksi mogla bit sve veća.

Bibliografija

[1] *DM-66 - Spatial Indexing*, https://gistbok.ucgis.org/bok-topics/spatial-indexing#Different, Accessed: 2021-06-28.

[2] *PostGIS History*, http://www.refractions.net/products/postgis/history/, Accessed: 2021-06-28.

[3] *Raster and Vector Data in GIS*, https://spatialvision.com.au/blog-raster-and-vector-data-in-gis/, Accessed: 2021-06-28.

[4] *Spatial Data Structures*, https://courses.cs.vt.edu/cs3114/Fall08/Quadtree.pdf, Accessed: 2021-06-28.

[5] Tor Bernhardsen, *Geographic information systems: an introduction*, John Wiley & Sons, 2002.

[6] Elisa Bertino, Beng Chin Ooi, Ron Sacks-Davis, Kian Lee Tan, Justin Zobel, Boris Shidlovsky i Daniele Andronico, *Indexing techniques for advanced database systems*, sv. 8, Springer Science & Business Media, 2012.

[7] Paul Bolstad, *GIS fundamentals: A first text on geographic information systems*, Eider (PressMinnesota), 2016.

[8] Open Geospatial Consortium et al., *OpenGIS Implementation Standard for Geographic information-Simple feature access-Part 2: SQL option*, Open Geospatial Consortium Inc (2010).

[9] Zdravko Galić, *Geoprostorne baze podataka*, (2006).

[10] Regina Hsu, *PostGIS in action*, Manning Publications, 2015.

[11] Jim Melton i Andrew Eisenberg, *SQL multimedia and application packages (SQL/MM)*, ACM Sigmod Record (2001).

Sažetak

U ovom diplomskim radu obrađena je tema prostornih baza podataka te prikazana mogućnost korištenja. U prvom poglavlju detaljno je objašnjeno što su to geoinformacijski sustavi. Zatim je u sljedećem poglavlju prikazano što su to prostorni podaci, koji modeli postoje za njihovu pohranu te kako se rade upiti i indeksiranje nad njima. U trećem poglavlju prikazan je objektno-orijentirani sustav za upravljanje prostornom bazom podataka, PostGIS. Pomoću njega su opisanje vrste podataka te funkcije za rad nad njima. Posljednje poglavlje sadrži stvarni primjer prostorne baze podataka. Prikazuje se korištenje prostornih baza za unapređenje web i mobilne aplikacije PayDo za plaćanje parkinga. Opisano je kreiranje baze i funkcija za rad s podacima te provođenje upita na bazu.

Summary

The main topic of this master thesis is spatial databases and it's possibility of usage. In the first chapter, it is explained in detail what geographic information system is. After that, it's described what the spatial data are, which models are used to storage them, how to set indexes on data and make queries on them. The next chapter explains an object-relational database system, PostGIS. Using this system, we describe all types of spatial data and functions for that data. The last chapter comprises a real example of spatial database. It shows the usage of spatial database for improvement of web and mobile application PayDo, which is used for parking payment. Also, this chapter describes how to create spatial database, functions for this data and queries.

Životopis

Rođena sam 15. ožujka 1996. godine u Dubrovniku. Osnovnu školu završila sam u Kuni na Pelješcu te potom upisala srednju Turističku i ugostiteljsku školu Dubrovnik, smjer hotelijersko-turistički tehničar u Dubrovniku. Nakon srednje škole, 2014. godine upisujem preddiplomski sveučilišni studij Matematika, smjer nastavnički na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta u Zagrebu. Isti završavam 2018. godine kada upisujem diplomski studij Računarstvo i matematika na istom odsjeku. Zapošljavam se u tvrtci Infoart 2020. godine.