SVEUČILIŠTE U ZAGREBU - GEODETSKI FAKULTET
UNIVERSITY OF ZAGREB - FACULTY OF GEODESY
Zavod za geomatiku; Katedra za geoinformatiku
Institute of Geomatics; Chair of Geoinformatics
Kačićeva 26; HR-10000 Zagreb, CROATIA
Web: www.geof.unizg.hr; Tel.: (+385 1) 46 39 222; Fax.: (+385 1) 48 28 081

Usmjerenje: Geoinformatika, Diplomski studij geodezije i geoinformatike

DIPLOMSKI RAD

Ispitivanje mogućnosti klasične baze prostornih podataka kod aplikacija za praćenje objekata u realnom vremenu

Izradio:

Ivan Kolar

Mala Erpenja 45

49217 Krapinske Toplice

ivkolar@geof.hr

Mentor: prof. dr. sc. Damir Medak

Zagreb, rujan 2014.

I. Autor
Ime i prezime: Ivan Kolar
Datum i mjesto rođenja: 10. 08. 1990., Zabok, Republika Hrvatska
II. Diplomski rad
Naslov: Ispitivanje mogućnosti klasične baze prostornih podataka kod aplikacija za praćenje objekata u realnom vremenu
Mentor: prof. dr. sc. Damir Medak
III. Ocjena i obrana
Datum zadavanja zadatka: 13. 01. 2014.
Datum obrane: 12. 09. 2014.
Sastav povjerenstva pred kojim je branjen diplomski rad: prof. dr. sc. Damir Medak dr. sc. Mario Miler prof. dr. sc. Drago Špoljarić

Zahvala:

Hvala obitelji na velikoj podršci i odricanjima kako bih došao do ovog koraka u mom životu, te prijateljima i kolegama koji se samnom vesele svakom uspjehu, pa tako i ovom.

Zahvaljujem mentoru prof. dr. sc. Damiru Medaku koji me je svojim pristupom na predavanjima, savjetima i idejama inspirirao da naučim više.

Hvala dr.sc. Mariu Mileru na velikom znanju, podršci i povjerenju koje mi je pružao za vrijeme studiranja te prilikom izrade ovog diplomskog rada.

Ispitivanje mogućnosti klasične baze prostornih podataka kod aplikacija za praćenje objekata u realnom vremenu

Sažetak:

Svijet u 21. stoljeću ima ogromnu potrebu za smještanjem informacija, robe i ljudi u prostorni kontekst. Sustavi za praćenje važan su faktor u svijetu u kojem je potreba za praćenjem objekata sve veća. Praktični je zadatak izraditi prototip za praćenje objekata realnom vremenu. U bazu prostornih podataka će se nakon nekog pohraniti velik dio prostornih informacija, što stvara potrebu za te testiranjem iste. Cilj je ispitati kako će se ponašati PostgreSQL/PostGIS klasična baza prostornih podataka kod masovnog unosa podataka te istražiti i ispitati njezine mogućnosti.

Ključne riječi: *sustav za praćenje, testiranje baze prostornih podataka, PostgreSQL / PostGIS*

Examining the possibilities of a classic spatial database with real-time object tracking applications

Abstract:

The world in the 21st century has a tremendous need to locate information, goods and people into a spatial context. Tracking systems are an important factor in a world where the need for monitoring objects grows. The practical task is to create a prototype for object tracking in real time. Spatial database will store a large amount of spatial information and there is a need to do a benchmark. The aim is to examine how it will behave with mass data entry and examine the possibilities of classical spatial database PostgreSQL / PostGIS's.

Keywords: *tracking system, spatial database benchmark, PostgreSQL / PostGIS*

S A D R Ž A J

1. Uvod

Poznato je da više od 80% informacija čine prostorne informacije. Svijet u 21. stoljeću ima ogromnu potrebu za smještanjem informacija, robe i ljudi u prostorni kontekst. Razvojem informacijske tehnologije, posebno one mobilne, polako su se stvarali uvjeti za sustave za praćenje objekata u realnom vremenu. Unazad nekoliko godina svjedoci smo novog ogromnog tržišta pametnih mobitela, tableta i ostale mobilne tehnologije. Tržište je to koje je vrlo zanimljivo i geodetskoj struci, a ponajviše geoinformatičkoj koja je napokon dobila uvjete za razvoj vlastitih ideja i realizaciju brojnih sustava koji se već danas smatraju dijelom svakodnevnog života. Razvijene su brojne aplikacije koje imaju upravo već spomenutu prostornu domenu. Sustavi za praćenje važan su faktor u svijetu u kojem je potreba za praćenjem objekata sve veća.

Na odabir upravo ove teme nadahnula me znatiželja da saznam što se nalazi u pozadini sustava za praćenje. Iako je krajnji rezultat takvih sustava za običnog korisnik tek jedna točka na karti, u pozadini se kriju brojne informacijsko-komunikacijske tehnologije.

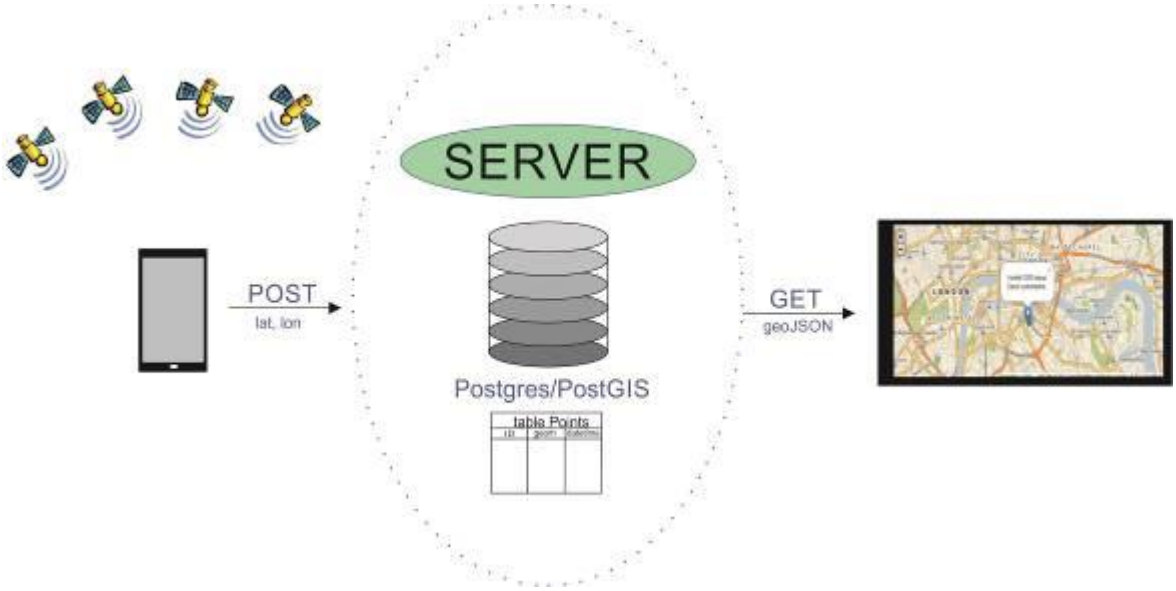
Praktični je zadatak izraditi prototip za praćenje objekata realnom vremenu. Ideja je započela od „Backitude: GPS Location Tracker“ Android aplikacije koja između ostalog ima mogućnost slanja trenutne lokacije mobilnog uređaja na proizvoljan server. Razvojem koncepta i izradom sustava za praćenje, došao sam do jednog važnog pitanja: Kako će sustav funkcionirati nakon nekog vremena? Pitanje je to koje se koncentrira na jednu komponentu sustava, a to je baza prostornih podataka u koju će se nakon nekog vremena pohraniti velika količina prostornih informacija. U tu svrhu služe testiranja za bazu podataka (*benchmark*) poput Vespa (W. Paton, 2000.) i Sequoia (Stonebraker,1993). Cilj je istražiti dosad izvršene testove, pronaći idealne za ovaj zadatak te ispitati kako će se ponašati klasična baza prostornih podataka kod masovnog unosa podataka te istražiti i ispitati mogućnosti klasične baze prostornih podataka Postgres/PostGIS-a.

1.1. Koncept

Izrada ovog rada započeta je postavljanjem problema i razvojem koncepta. U prvom dijelu započinje se sa izradom sustava za praćenje korisnika u realnom vremenu. Potrebno je izraditi protokol te odabrati tehnologije će se koristiti. Rezultat takvog sustava za krajnjeg korisnika je lokacija objekta na digitalnoj karti. Pogledom unaprijed dolazimo do novih pitanja i problema koje je potrebno riješiti.

1.1.1. Izrada prototipa za praćenje lokacije korisnika u realnom vremenu

Protokol izrade sustava započinje razmatranjem od kojih će se komponenata sastojati sustav. Prva od njih svakako je mobilni uređaj koji ima integriranu GPS jedinicu i pristup Internetu putem mobilne mreže ili WLAN-a (*Wireless local area network*). Mobilni uređaj ima zadatak da prosljeđuje svoju lokaciju na server. Druga je komponenta server sa bazom podataka u koju će se pohranjivati lokacije objekta. Treća je komponenta mogućnost korisnika da sazna lokaciju objekta. Jedan od standardnih načina za razmjenu prostornih informacija je geoJSON format. Okvirni koncept sustava prikazan je na slici 1.



Slika 1: Okvirni koncept sustava

1.1.2. Prikaz lokacije objekta na karti

Pohranjene koordinate u bazi nemaju smisla ako se ne prikažu u kombinaciji s nekom digitalnom kartom. Ovaj dio zamišljen je kao web gis koji će kao podlogu koristiti Openstreetmap te na njoj prikazati lokaciju korisnika. Lokacija je predstavljena markerom koji se svake sekunde osvježava te prikazuje lokaciju korisnika u realnom vremenu.

1.1.3. Ispitivanje baze prostornih podataka

Svaki sustav za praćenje mora odgovoriti na pitanje: gdje je bio objekt u određenom trenutku? Ako takav sustav radi konstantno te teoretski bilježi lokacije više korisnika kroz teoretski neograničeno vrijeme, postavlja se pitanje: u kojem će trenutku baza podataka početi pokazivati svoje manjkavosti? Predikcija takvog slučaja u relativno kratkom vremenu moguća je korištenjem postojećih zapisa. Naime, postoji mnogo dostupnih prostornih informacija koje mogu biti korištene za testiranje. Ideja je da se postupnim povećavanjem količine podataka prati funkcionalnost rada baze, a samim time i sustava za praćenje. Osim količine, posebnu pozornost posvećena je načinu zapisa prostornih informaciju u bazu u cilju saznavanja najoptimalnijeg i najekonomičnijeg načina.

2. Prethodna istraživanja

Ispitivanja mogućnosti baza podataka („*benchmark*“) koriste se kako bi se ispitala funkcionalnost i učinak sustava. Kod funkcionalnosti se traži odgovor na pitanje: „Što sustav može napraviti?“, a ispitivanja učinka traže odgovor na pitanje: „Koliko vremena i prostora treba sustavu da izvrši neki zadatak?“ (Paton i sur., 2000). U ovom radu korištena su ispitivanja učinka.

Trenutno ne postoji jedinstven okvir za ispitivanje učinka baza prostornih podataka. Dok neprostorne baze podataka već koriste standardizirane testove poput TPC–ovih (*The Transaction Processing Performance Council*), one s prostornom komponentom zaostaju. Dio problema leži u činjenici da je teško razviti okvir koji bi sve sustave testirao na jednak i pošten način zbog različitih načina realizacija sustava. S druge strane, neki sustavi nemaju implementirane određene funkcije, pa je njihovo testiranje učinka nemoguće. Testovi prostornih baza podataka ovise i o tipovima podataka koje pohranjujemo. Oni mogu biti vektorski i rasterski, pa u skladu s time su i različita ispitivanja.

U nastavku je objašnjen pristup nekoliko razvijenih ispitivanja i primjera za pojedinačne slučajeve koje su korištene za kreiranje vlastitog okvira za ispitivanje učinka baza prostornih podataka.

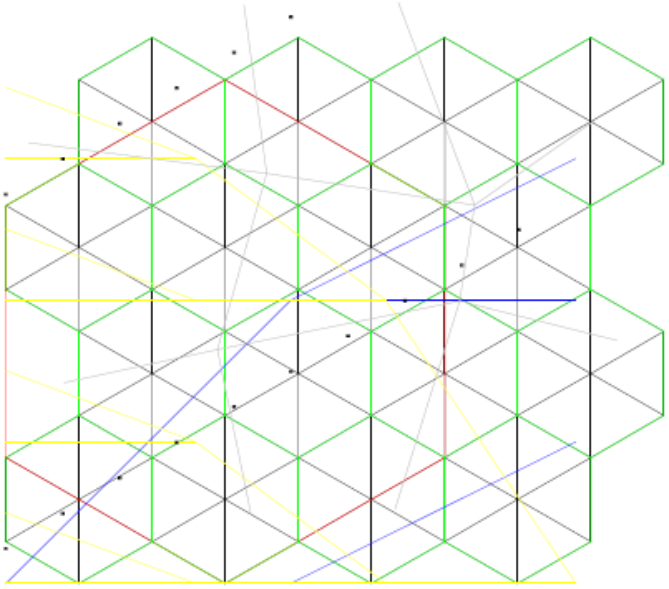
Paton i suradnici (2000.) uspoređuju Postgres i Rock&Roll objektno-orijentirane baze podataka. Autori su postavili tri zahtijeva koji bi trebali biti zadovoljeni:

1. Jednostavnost. Implementacija struktura podataka i upiti trebali bi biti realizirani u kratkom roku s jednostavnim načinom nabave setova podataka.

2. Široka uporabljivost. Ispitivanje bi trebalo biti omogućeno na čim više tipova prostornih podataka.

3. Prilagodljivost. Ovisno o kontekstu, testiranje treba omogućiti unos čim manje ili čim veće količine podataka kako bi se izvršile analize učinka.

U tu svrhu koriste VESPA (*a Benchmark for Vector Spatial Databases*) test za ispitivanje baza prostornih podataka za unos vektorskih podataka. Radi se o testu koji ne koristi stvarne podatke, već umjetne podatke prvenstveno radi lakše nabave takvog seta podataka, kontrole veličine i oblika prostornih objekata te predviđanja rezultata upita. Na slici 2 vidljiv je uzorak koji je korišten za testiranje. Uzorak sadrži različite tipove prostornih podataka (točke, linije, poligoni)



Slika 2: Uzorak za testiranje (Paton, 2000.)

Testovi su kategorizirani. Između ostalog, mjeri se vrijeme unosa uzorka u bazu i vrijeme brisanja određenog zapisa, testiranje pomoću topoloških odnosa, metričke operacije poput udaljenosti i površina, spajanje objekata i sl.

Testovi su izvedeni u obje baze podataka, a rezultati su dani u vremenu u ovisnosti od količine podataka.

Stonebraker i suradnici (1993.) koristili su SEQUOIA 2000 test. Prednost mu leži u činjenici da koristi stvarne podatke. Međutim, radi se o testiranju baza podataka za unos i analize rasterskih podataka.

Ray, Simion i Brown (2011.) izradili su test za vektorske tipove podataka koji koristi stvarne podatke nazvan *Jackpine*. Cilj im je bio da test bude primjenjiv na velikoj količini tipova geometrijskih podataka, funkcionalan sa stvarnim setom podataka, proširiv i mobilan. Ispitivanje su izveli na PostgreSQL, MySQL i Informix-u, i to u dva segmenta: mikro i makro ispitivanje. Mikro testiranje obuhvaća topološke upite pridruživanja (JOIN) na temelju proširenog modela 9 presjeka, analitičkih funkcija (udaljenost, convex hull, envelope, buffer) te agregacijskih funkcija (najdulja linija, najveća površina, ukupna duljina i ukupna površina). Makro testiranje koristi složenije testove iz stvarnosti poput geokodiranja, pretraživanja karata, analiza rizika od poplava i sl. Krajnji rezultat testiranja dan je u obliku grafova te ukupnoj ocjeni.

Sličan test izveo je R. Power (2009.) koji je osim topoloških testova na stvarnim podacima mjerio vrijeme jednostavnih upita unutar MySQL-a i PostgreSQL-a kao što su brojanje redaka u tablici i traženje najmanjeg pravokutnika koji okružuje geometriju.

R. Suter (2012.) izveo je ispitivanje učinka uspoređujući pritom MongoDB i PostgreSQL. Svoje testiranje podijelio je u tri dijela: ispitivanje vremena unosa, pretraživanja i kreiranja najmanjeg pravokutnika koji okružuje geometriju.

Rezultati ispitivanja također su dani u ovisnosti o količinu podataka. Ono što je zanimljivo u tom radu je ispitivanje upita sa i bez prostornog indeksa.

Navedena prethodna istraživanja bila su orijentirana na usporedbu dvaju ili više baza podataka. Korišteni su razni testovi čiji su rezultati prikazali učinkovitost pojedinih baza podataka. U ovom radu fokus će biti na jednoj bazi prostornih podataka, a testirat će se njezine mogućnosti za korištenje u sustavu za praćenje objekata u realnom vremenu.

3. Materijali i metode

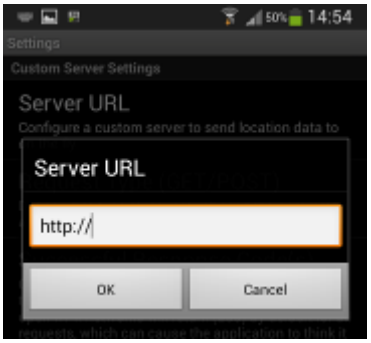
I ovom poglavlju opisan je postupak izrade prototipa sustava za praćenje objekta u realnom vremenu i način rješavanja nastalih problema. Prethodno su opisane tehnologije koje su pritom korištene.

3.1. Korištene tehnologije

Prototip sustava izrađen je kombinacijom više tehnologija. Opisane su njihove mogućnosti, prednosti i nedostaci, kao i značajke bitne za ovaj rad.

3.1.1. Backitude: GPS Location Tracker

Backitude je jednostavna i besplatna Android aplikacija (za 2.2 i novije verzije) za dohvat lokacije mobilnog uređaja u proizvoljnom vremenskom intervalu. Ne zauzima puno memorije i ne utječe znatno na pražnjenje baterije. Aplikacija omogućuje slanje, sinkronizaciju ili izvoz zabilježenih položaja korisnika u određeni sustav za kartiranje. Jedna od važnijih mogućnosti je dijeljenje podataka o položaju. Do kolovoza 2013. godine Backitude je imao mogućnosti sinkronizacije lokacija s Googleom što je pridonijelo poboljšanju točnosti prilikom pružanja usluge za određivanje položaja. Točnije, tada je to bila jedina svrha te aplikacije. Gašenjem Google Latitude API-ja, Backitude je doživio redizajn. Korisnici tako imaju niz novih mogućnosti. Jedna od njih je definiranje proizvoljnog url-a servera kojem želimo proslijediti koordinate lokacije korisnika (slika 3), ili pohrana u KML ili CSV datoteku za korištenje lokacija u nekom od sustava za kartiranje.



Slika 3: Definiranje url-a servera

Backitude za pozicioniranje koristi GPS, lokacije Wi-Fi mreža i lokacije dobivene triangulacijom repetitora mobilnih mreža. Navedene tehnike mogu se koristiti same

ili u kombinaciji te gotovo neprimjetno rade u pozadini uređaja. Moguća je kontrola učestalosti ažuriranja lokacije i ograničenja točnosti. Prilikom nestanka signala koji se koristi za lociranje, koristi se bezmrežna pohrana koja pamti zadnju lokaciju. Kod ponovne uspostave signala, podaci se sinkroniziraju.

Ograničenja za minimalnu točnost i dodatne opcije osiguravaju da se generiraju samo lokacije koje su prihvatljive u pogledu točnosti. Aplikacija također ima algoritme koje prepoznaju kada nema smisla čekati dovoljno pouzdanu i točnu lokaciju, što značajno doprinosi trajanju baterije mobilnog uređaja (url 2).

3.1.2. Postgis

PostGIS je proširenje PostgreSQL-a, objektno – relacijske baze podataka za prostorne podatke. Proširenje podržava prostorne SQL upite za geometrijske i geografske objekte.

Podržava sve vektorske tipove GIS objekata koji su definirani u „*Simple Features for SQL 1.2.1.*“ standardu od *OpenGIS Consortium* (OGC) i ISO *„SQL/MM Part 3: Spatial“* dokumentu. PostGIS također podržava rasterske tipove podataka i topološki model.

OGC i ISO standardi definiraju 2D (x/y), 3D (x/y/z, x/y/m) and 4D (x/y/z/m) varijante točaka, linija, poligona, zakrivljenih entiteta, poliedra i TIN-a (*Triangulated Irregular Network*). OGC i ISO specifikacije definiraju tekstualne i binarne zapise geometrijskih objekata: *Well Known Text* (WKT) i *Well Known Binary* (WKB).

Obje vrste zapisa uključuju informacije o tipu i koordinatama točaka koje formiraju objekt (url 3).

Primjeri tekstualnih zapisa (WKT) prostornih objekata:

POINT(32.01 34.01)

LINESTRING(0 1,4 1,5 4)

POLYGON((0 1,4 2,4 4,0 6,0 1))

MULTIPOINT((5 0),(1 6))

MULTILINESTRING((0 1,4 1,4 3),(1 2,3 4,5 6))

MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)), ((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))

GEOMETRYCOLLECTION(POINT(2 3),LINESTRING(2 3,3 4))

Primjer WKB zapisa točke:

0101000020E6100000E17A14AE47014040E17A14AE47014140

Prethodni WKB zapis u heksadecimalnom zapisu nije interni format PostGIS-a, već samo njegova prezentacija. Interna struktura opisana je pomoću *gserialized* formata koji je naslijedio *serialized_lwgeom* format.

Interni format PostGIS-a sastoji se od zaglavlja i geometrije:

Zaglavlje

<size> veličina korištena od PgSQL
<srid 3 bajta
flags> 1 bajt
<bbox-xmin> minimalni pravokutnik (opcionalno)
<bbox-xmax>
<bbox-ymin>
<bbox-ymax>

Geometrija

<pointype> Točkasti tip
<npoints> Ako je prazan, prima vrijednost 0, u protivnom 1
[double] Prva koordinata
[double] Druga koordinata
[double] Treća koordinata

<linestringtype> Linijski tip
<npoints> Ako je prazan, prima vrijednost 0, u protivnom 1
[double] Točke koje definiraju liniju
...
[double]

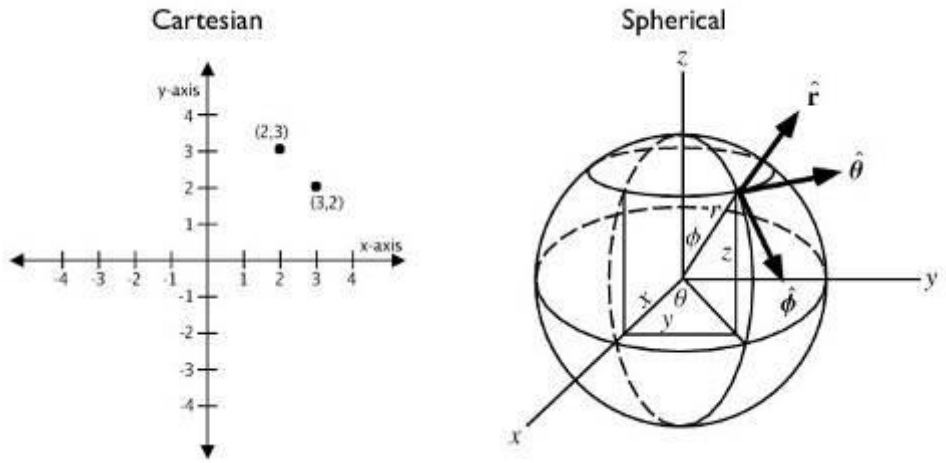
OpenGIS specifikacije zahtijevaju da prostorni objekti imaju unutarnje pohranjen format koji uključuje identifikator referentnog prostornog sustava (SRID). SRID je potreban prilikom kreiranja prostornih objekata za unos u bazu podataka.

U tu je svrhu PostGIS kreirao *spatial_ref_sys* tablicu sukladnu s OGC zahtjevima. Ona sadrži preko 3000 poznatih referentnih prostornih sustava i parametre za transformaciju. Ukoliko referentni sustav nije sadržan u tablici, moguće ga je kreirati pomoću proj4 strukture (url 4).

Prostorne informacije mogu se zapisati u bazu kao geometrijski ili kao geografski tip. U radu će se ispitivati razlika u veličini tablica tih tipova, kao i veličine prostornih indeksa i toast tablica vezanih uz te tablice. U idućim potpoglavljima objašnjeni su geometrijski i geografski tipovi, prostorni indeksi u PostGIS-u kao i toast tablice.

3.1.2.1 Geometrijski i geografski tip

PostGIS koristi dva načina zapisa prostornih tipova podataka. Najčešće se koristi geometrijski (eng. *geometry*). U tom slučaju sve se operacije izvršavaju na ravnini, bez obzira jesmo li definirali referentni sustav u projekciji ili na elipsoidu. Zbog toga je najkraća udaljenost između dvije točke u tom slučaju dužina. To znači da se računanja s geometrijom izvršavaju u Kartezijevom koordinatnom sustavu (url 5).



Slika 4: Usporedba koordinata položaja u ravnini i sferi (url 5)

Korištenjem geografskom zapisa riješen je problem netočnih udaljenosti na elipsoidu ili sferi (slika 4). Upravo je sfera osnova za računanje najkraće udaljenosti između dviju točaka, a rezultat je kružni luk u metrima (za razliku od geometrije koja kao mjernu jedinicu koristi vlastitu jedinicu). Međutim, računanja na sferi koriste složenije matematičke operacije, a ukoliko u obzir uzmemo i elipsoid kao osnovu, dolazimo do još većih komplikacija i trajnijih izračuna. Složenija računanja za geografiju utjecala su i na broj funkcija koji je značajno manji nego kod geometrijskog zapisa. Za razliku od geometrije koja podržava velik broj referentnih sustava sa i bez projekcije, geografski zapisi zasad podržavaju samo WGS84 elipsoid (url 6).

Ključno pitanje je: kako odabrati način zapisa prostornih podataka? PostGIS u svojoj dokumentaciji (url 7) predlaže iduće:

1. Ako su prostorni podaci sadržani u malom području, najbolje je odabrati odgovarajuću projekciju i koristiti geometrijski zapis.

2. Ako se radi o globalnim podacima ili o podacima na razini kontinenata, poželjno je koristiti geografski zapis. Primjer je na slici 5.

3. Ako se radi o jednostavnijim operacijama nad podacima na većem području s minimalnim znanjem o projekcijama i spremni smo prihvatiti ograničen broj funkcija, dovoljno je koristiti geografski zapis.

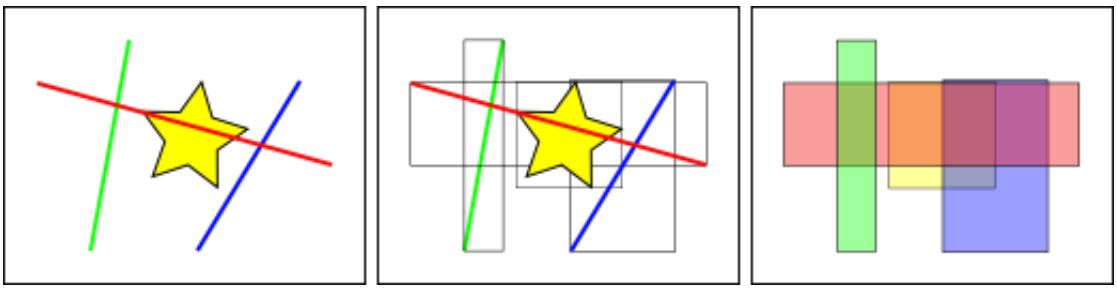
Slika 5: Primjer razlika u udaljenosti između Los Angelesa i Pariza (url 1)

Ukoliko se koristi geografski zapis, potrebno je dobro razmisliti o odnosu brzine izvršavanja zadataka i točnosti. Brzina rješavanja problema kod geografskog zapisa nad većim područjem manja je u odnosu na geometrijski zapis, ali je pritom točnost rezultata veća.

3.1.2.2 PostGIS prostorni indeksi

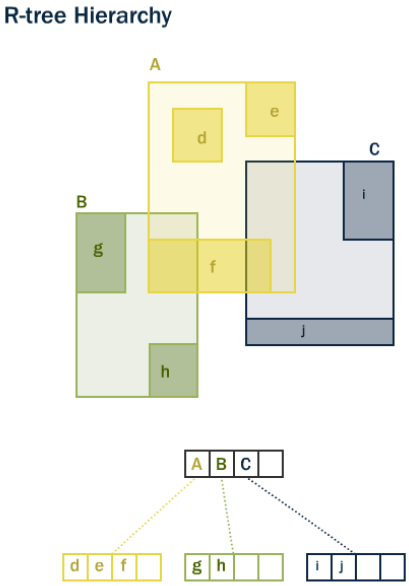
Prostorni indeksi jedni su od najvažnijih značajki baza prostornih podataka. Bitni su za korištenje pri velikoj količini pohranjenih podataka, što je bitno i za sustav za praćenje koji pohranjuje velik broj položaja objekta. Bez indeksa, traženje pojedinog objekta zahtijeva sekvencijalno skeniranje svakog zapisa u bazi. Indeksiranjem se pretraga ubrzava na način da se podaci organiziraju u stabla za pretraživanje (url 8).

Neprostorni indeksi kreiraju hijerarhično stablo koje se temelji na vrijednosti stupca koji se indeksira. Prostorni indeksi nemaju mogućnosti da označe same objekte, već se indeksiraju najmanji pravokutnici koji okružuju pojedini objekt. Na slici 6 prikazan je primjer indeksiranja. Prostorni indeksi traže najveću i najmanju vrijednost po osima te se kreira najmanji pravokutnik na temelju tih vrijednosti (url 8). Kod prostornog pretraživanja uspoređuju se topološki odnosi između najmanjih pravokutnika. Presjek tih pravokutnika nužan je, ali ne i dovoljan uvjet da se dva objekta sijeku. Primjer je odnos na slici 6 između žute zvijezde i plave dužine. Njihovi najmanji pravokutnici se sijeku, ali sami objekti se ne sijeku. Iako u prvom koraku nije dalo rješenje, takav princip uvelike smanjuje broj računanja.



Slika 6: Princip indeksiranja prostornih objekata (url 8)

PostGIS koristi R-stablo za strukturu indeksa. R-stabla podijele podatke u pravokutnike, pravokutnike unutar pravokutnika itd. Takva struktura sama održava gustoću podataka i veličinu objekata.



Slika 7: Hijerarhija R-stabla (url 8)

GiST (Generalized Search Tree) je općeniti tip indeksa koji uključuje R-stabla, B+ stabla, hB-stabla i mnoge druge tipove, što eliminira potrebu za kreiranjem višestrukih indeksa za rukovanje u različitim primjenama.

GiST kao jedinstven indeks ima važnu prednost pred ostalim indeksima: proširivost podataka i upita. Prethodni indeksi su prošrivi jedino u kontekstu podataka kojim rukuju. Primjerice, Postgres B+ stabla koriste operatore usporedbe (<,>,=), a R-stabla upite poput sadržavanja, jednakosti. GiST s druge strane ujedinjuje različite operatore.

GiST ima strukturu koja je balansirana kao B-stablo i sadrži parove ključa i pokazivača. Za razliku od B-stabla, ključevi nisu brojčanog tipa, već je ključ član klase koju definira korisnik te predstavlja svojstva koja su ista za sve podatke koji se mogu dohvatiti sa pokazivačem koji je vezan za ključ. Na primjer, ključevi u B+ stablu su skupovi brojeva, u R-stablu su najmanji pravokutnici, a u GiST-u su skupovi i podskupovi podataka. Da bi GiST radio, treba razmisliti što predstaviti u ključevima i tada napisati četiri metode za klasu od ključa. Te četiri metode su konzistentnost, unija, *Penalty* i *PickSplit* (url 9).

3.1.2.3 Toast tablice

Za razumijevanje principa toast tablica i razloga njihova nastajanja, potrebno je objasniti kako Postgres pohranjuje podatke.

Svaka tablica i indeks pohranjeni su u niz stranica (*eng. pages*) koje standardno imaju veličinu 8kB, iako se ta vrijednost može promijeniti. Sve su stranice logički ekvivalentne, što znači da pojedini zapis može biti pohranjen na bilo kojoj stranici. Kod indeksa je prva stranica rezervirana kao metastranica koja sadrži informacije o upravljanju indeksom. Postoji više tipova stranica unutar indeksa, ovisno o tome koje metodu pristupa indeks koristi (url 22).

Veličina od 8kB jednaka je vrijednosti 8192 bajta. Prvih 24 bajta svake stranice rezervirano je za zaglavlje stranice (*PageHeaderData*). Zaglavlje sadrži općenite informacije o stranici, uključujući pokazivače na slobodna mjesta na stranici. Poslije njega slijede parovi odmaka (*offset*) i duljine (*length*) koji pokazuju na stvarni zapis. Zapis u ovom kontekstu odgovara jednom redku u tablici. Svaki par zauzima 4 bajta po zapisu. Zatim slijede slobodna mjesta za pokazivače koji se smještaju od početka i za zapise na smještaju od kraja. Posljednji dio stranice je rezerviran za indeksove metode pristupa specifičnim podacima.

Budući da Postgres koristi fiksnu veličinu stranice i ne dozvoljava pojedinom zapisu da se proširi na drugu stranicu, nemoguće je direktno pohraniti zapise s velikim atributima. Prije verzije Postgres-a 7.1 je zbog toga postojalo ograničenje veličine jednog reda, tj. zapisa. Navedena i sve novije verzije podržavaju kompresiranje i razbijanje redaka u više fizičkih redaka. Ta metoda nazvana je TOAST (*The Oversized-Attribute Storage Technique*). Toast tablice rezultat su korištenja navedene tehnike. Prema rezultatu testiranja od strane Postgres-a na kojem su bile pohranjivane html stranice i njihovi url-ovi, glavne tablice sadržavale su oko 10% od ukupne veličine podataka (url 23). Pritom je važno naglasiti da taj udio ovisi i podacima.

3.1.3. Python programski jezik i proširenja

Python je objektno orijentiran viši programski jezik s dinamičnom semantikom. Podržava module i pakete i nema kompajliranja. Za izradu ovom rada korišten je s dodacima poput flaska, psycopg2

Flask je framework za web aplikacije napisan u Python programskom jeziku i temelji se na Werkzeug WSGI alatu i Jinja2.

Pomoću fleksibilnog Python programskog jezika pruža jednostavan predložak za razvoj web aplikacija. Uz njegovo ime često se koristi naziv *microframework* jer je njegova jezgra jednostavna, ali proširiva. Nema potrebe za apstraktnim slojem baze podataka, forme za provjeru valjanosti ili bilo koje druge komponente gdje već postoje dodatne biblioteke da pruže funkcionalnost. Međutim, Flask pruža dodatke koji povećavaju funkcionalnost u aplikaciji kao da su već implementirani u Flask (url 10).

Psycopg2 je dodatak Python-u za spajanje na PostgreSGL bazu podataka. Njegove karakteristike su da je to malen i brz, ali i jako stabilan dodatak. Za razliku od ostalih sličnih dodataka, Psycopg2 je pogodan za aplikacije koje kreiraju i poništavaju mnogo kursora i koje koriste velik broj insert i update naredbi. Moguće ga je kompajlirati i pokrenuti na Linux, FreeBSD, Solaris, MacOS X i Windows arhitekturi. Funkcionira s Pythonovom verzijom 2.4 i više te PostgreSQL 7.4 i višim verzijama (url 11).

3.1.4. Javascript i jQuery

Javascript (JS) je dinamički programski jezik. Koristi se za interaktivnost web stranica te sa strane klijenta omogućuje kontrolu preglednika, asinkronu komunikaciju i promjenu prikazanog sadržaja. Sa serverske strane koristi se za izradu stolnih i mobilnih aplikacija, igra i sl.

JQuery Javascript biblioteka omogućuje jednostavno obuhvaćanje i rukovanje HTML dokumentom. Također, pojednostavljuje upravljanje eventovima, animacijama te korištenje Ajax-a pomoću API-ja koji radi na velikom broju preglednika. Dodatne prednosti su mu mogućnosti proširivanja, što je privuklo mnoge korisnike koji koriste Javascript (url 12).

3.1.5. Leaflet API

Leaflet je moderna javascript biblioteka otvorenog koda za izradu interaktivnih karata koje se mogu koristiti i na mobilnih uređajima. Razvijena je od Vladimira Agafonkina s timom suradnika. Iako je biblioteka velika svega 33 KB, sadrži većinu potrebnih značajki za izradu online karata.

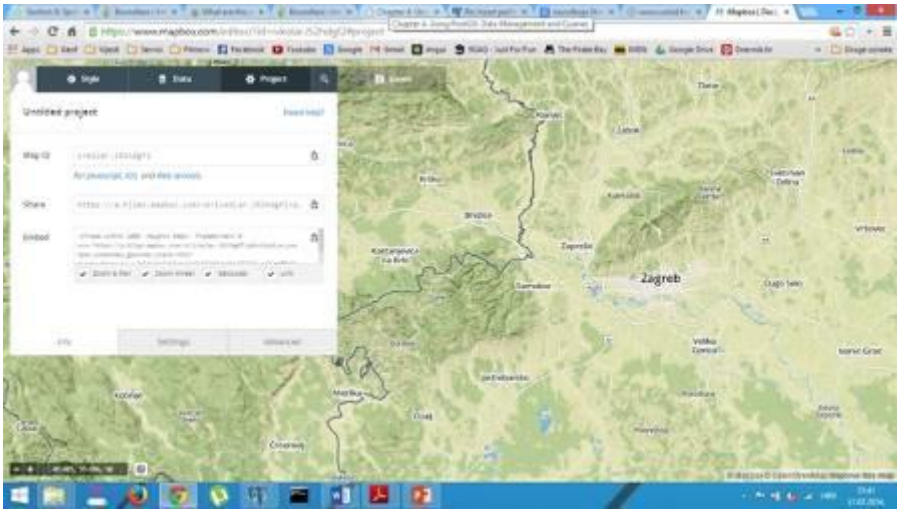
Dizajniran je na način da bude jednostavan, a da pritom ne gubi na performansama i uporabljivosti. Radi na svim glavnim stolnim i mobilnim platformama. Uzimajući u obzir prednosti HTML5 i CSS3 na novijim preglednicima, istovremeno radi i na starijim verzijama. Može se dodatno proširiti velikim brojem razvijenih proširenja, a čitljiv izvorni kod otvara prostor daljnjem razvoju biblioteke. Primjer jednostavnog sučelja prikazan je na slici 8 (url 13).

Slika 8: Primjer web gis sučelja izrađenog pomoću leafleta

3.1.6. Mapbox

Mapbox je cloud platforma za dijeljenje brzih, ljepih i interaktivnih web karata. Dostupne su brojne javno dostupne karte, a moguće je i dizajnirati vlastitu kartu koristeći TileMill. Korišten je u brojnim i poznatim servisima kao što su Foursquare, Pinterest ili Evernote (url 14).

U Leaflet API-ju jednostavno je dodati kartu koja je definirana u Mapboxu jer svaka kreirana karta dobiva jedinstven identifikator (Map ID). Na slici 7 prikazan je dio izrade karte. Svaka se karta sprema kao zaseban projekt kojem je kasnije moguće pristupiti i modificirati. U slučaju nekih izmjena na projektu, promjena karte na sustavu koja koristi kartu sa Mapboxa izvršava se u kratkom roku.



Slika 9: Izrada karte s Mapbox-om

3.2. Izrada sustava za praćenje objekata

Praktični dio diplomskog rada je izrada prototipa sustava za praćenje objekata. Prema već opisanom konceptu, izrada je podijeljena u tri koraka koja su opisana u idućim poglavljima.

3.2.1. Aplikacija za dohvat lokacija i pohranu podataka u bazu

Kreiranje aplikacije započeto je sa *Flask* klasom te kreiranjem instance te klase koja će biti WSGI aplikacija. Prvi argument je ime modula ili paketa aplikacije. Budući da je u mojem slučaju jedan modul, koristim *__name__* jer ovisno o tome je li pokrenuta kao aplikacija ili uvezen poput modula naziv će biti drugačiji. To je potrebno definirati kako bi Flask znao gdje će potražiti predloške, statične datoteke i slično (url 15).

app = Flask(__name__, static_url_path='')

Aplikacija se spaja na PostgreSQL/PostGIS bazu podataka te je bitno da kod svakog zahtjeva aplikacije ta veza funkcionira. Stoga je potrebno osigurati da se prije zahtjeva otvori veza s bazom, a nakon zahtjeva da se konekcija zatvori. Kreirana su tri *view*-a.

Prvi je *view* s POST metodom koja ima ulogu da proslijedi informacije o lokaciji korisnika serveru. U tom se koraku vrijednosti geografskih dužina i širina pridružuju zasebnim varijablama.
Za izradu *web* aplikacija ključno je reagirati na podatke koje klijent šalje serveru. U Flasku je ta informacija pružana pomoću globalnog *request* objekta. Primjenom *form* atributa imamo dostup formi primljenog objekta. Na taj način izvučene su vrijednosti geografskih širina i duljina.
Kako bi te vrijednosti pohranili u odgovarajuće mjesto u bazi, kreiran je pokazivač koji prvo omogućuje kreiranje odgovorajućeg SQL upita, a zatim i njegovo izvršavanje. Svaku promjenu u bazi potrebno je potvrditi sa commit metodom. Nakon pohrane, potrebno je zatvoriti vezu s bazom podataka.

Drugi view *getpoint* koristi GET metodu pošto izvlačimo podatke iz baze podataka. Kao i kod unosa podataka u bazu, i kod odabira podataka (*select*) potrebno je

kreirati pokazivač koji će omogućiti izvoz odgovarajućih podataka iz baze. U ovom slučaju radi se o točki koja je posljednja unesena u bazu. U skladu s konceptom, točka se izvozi u geoJSON format. Međutim rezultat takvog izvoza nije geoJSON objekt već string kod kojeg su se pojavile zagrade koje nisu izvorno dio geoJSON formata, naprimjer:

[('{"type":"Point","coordinates":[46.1021575927734,15.8014087677002]}',)]

Tekstualnoj varijabli (*result*) oduzeo sam prvi tri i zadnja četiri znaka kako bi zadovoljila strukturu geoJSON formata.

>>>>mypoint = str(result)[3:][:-4]

{"type":"Point","coordinates":[46.1021575927734,15.8014087677002]}

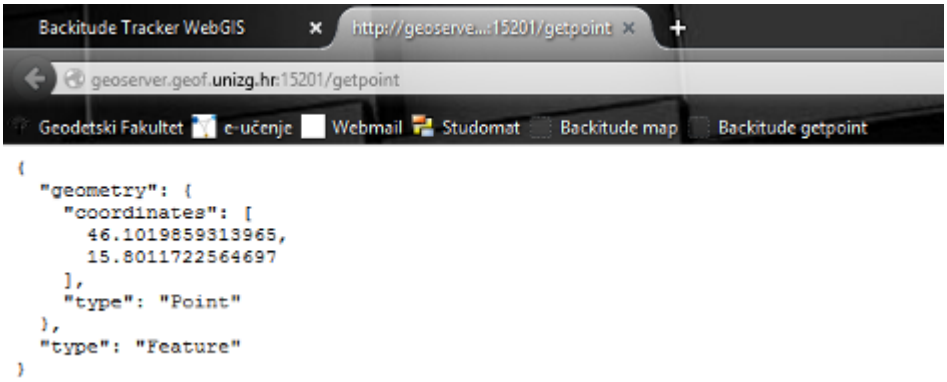
Rezultat prethodne modifikacije daje string koji ima izgleda kao geoJSON format, ali nije definiran kao takav, tj. nije geoJSON objekt. Dodatno, prethodni rezultat je samo dio prave strukture geoJSON formata, tj. njegova geometrija (url 16). Primjer geoJSON objekta za točku:

```
{
    "type": "Feature",
    "geometry": {
      "type": "Point",
      "coordinates": [46, 16]  },
    "properties": {
      "prop0": "value0"  }
}
```

Da bih taj string pretvorio u stvarni geoJSON objekt, koristio sam *jsonify()* metodu koja pretvara string u geoJSON objekt te dodaje ostale potrebne dijelove strukture (url 17).

mygeojson=jsonify({"type":"Feature","geometry":json.loads(mypoint)})

Rezultat drugog view-a vidljiv je na slici 10.



Slika 10: Rezultat drugog view-a koji vraća položaj točke kao geoJSON objekt

Treći i posljednji view sadrži funkciju koja vraća kartu, tj html datoteku.

Render_template metoda vraća predložak, tj html datoteku. Navedeni predložak opisan je u poglavlju vizualizacije objekta. Flask će predloške potražiti u *templates* mapi. Budući da je ovdje riječ o modulu, *templates* mapa nalazi se uz modul.

Struktura izgleda ovako:

/btwebgis.py
/templates
/map.html

Pritom *btwebgis.py* predstavlja web aplikaciju, *templates* je mapa s predlošcima, a *map.html* je predložak koji će poslužiti za vizualizaciju lokacije korisnika na karti.

Pokretanje aplikacije

```
ivkolar@geoserver:~$ cd diplomski
ivkolar@geoserver:~/diplomski$ source bin/activate
(diplomski)ivkolar@geoserver:~/diplomski$ python btwebgis.py
 * Running on http://0.0.0.0:15201/
 * Restarting with reloader
longitude: 15.801172256469727 latitude: 46.101985931396484
31.217.2.80 - - [02/Aug/2014 12:23:25] "POST / HTTP/1.1" 201 -
31.217.2.80 - - [02/Aug/2014 12:23:29] "GET /map HTTP/1.1" 200 -
31.217.2.80 - - [02/Aug/2014 12:23:32] "GET /getpoint HTTP/1.1" 200 -
31.217.2.80 - - [02/Aug/2014 12:23:34] "GET /getpoint HTTP/1.1" 200 -
31.217.2.80 - - [02/Aug/2014 12:23:36] "GET /getpoint HTTP/1.1" 200 -
31.217.2.80 - - [02/Aug/2014 12:23:38] "GET /getpoint HTTP/1.1" 200 -
31.217.2.80 - - [02/Aug/2014 12:23:40] "GET /getpoint HTTP/1.1" 200 -
longitude: 15.801172256469727 latitude: 46.101985931396484
31.217.2.80 - - [02/Aug/2014 12:23:41] "POST / HTTP/1.1" 201 -
31.217.2.80 - - [02/Aug/2014 12:23:42] "GET /getpoint HTTP/1.1" 200 -
31.217.2.80 - - [02/Aug/2014 12:23:44] "GET /getpoint HTTP/1.1" 200 -
31.217.2.80 - - [02/Aug/2014 12:23:46] "GET /getpoint HTTP/1.1" 200 -
31.217.2.80 - - [02/Aug/2014 12:23:48] "GET /getpoint HTTP/1.1" 200 -
31.217.2.80 - - [02/Aug/2014 12:23:50] "GET /getpoint HTTP/1.1" 200 -
31.217.2.80 - - [02/Aug/2014 12:23:52] "GET /getpoint HTTP/1.1" 200 -
31.217.2.80 - - [02/Aug/2014 12:23:54] "GET /getpoint HTTP/1.1" 200 -
longitude: 15.801172256469727 latitude: 46.101985931396484
31.217.2.80 - - [02/Aug/2014 12:23:56] "POST / HTTP/1.1" 201 -
```

Aplikacija je pokrenuta na serveru pomoću Putty-ja. Radi lakšeg praćenja i kontrole ispisane su koordinate svakog položaja koji je dohvaćen s mobitela (POST metoda). Položaj se dohvaćao ovisno o parametrima koji su podešeni na mobitelu (svake sekunde), ali zbog sporijeg pristupa Internetu i performansi mobitela znalo je doći do kašnjenja. Zbog toga je GET metoda koja izvlači koordinate iz baze prilagođena na dvije sekunde.

3.2.2. Baza prostornih podataka

Tablica u bazi podataka zamišljena je da odgovara zahtjevima sustava za praćenje. Sustav mora odgovoriti na pitanje gdje se objekt nalazio u određenom trenutku. Također, svaki taj događaj mora biti jedinstven pa svaki događaj ima jedinstveni ID. Druga je stupac vremenska oznaka kako bi znao na koji se trenutak u prostornoj komponenti zabilježio događaj. Treća i najbitnija komponenta je ona geometrijska. Zapisi u bazi podataka su točke (*Points*) s pripadnim geografskim širinama i duljinama. U bazu su pohranjeni u *Well Known Binary* (WKB) formatu koji nije čitljiv, ali zato nudi bolje performanse u odnosu na *Well Known Text* (WKT). Prikaz Postgres/PostGIS tablice prikazan je na slici 11.

id serial	the_time text	geom geometry(Point)
213	2014-06-29 09:34:10	0101000000000000
214	2014-06-29 09:34:27	0101000000000000
215	2014-06-29 09:36:05	0101000000000000
216	2014-06-29 09:36:21	0101000000000000
217	2014-06-29 09:36:39	0101000000000000
218	2014-06-29 09:36:56	0101000000000000
219	2014-06-29 09:37:13	0101000000000000
220	2014-06-29 09:40:08	0101000000000000
221	2014-06-29 09:40:25	0101000000000000
222	2014-06-29 09:40:42	0101000000000000
223	2014-06-29 09:40:58	0101000000000000
224	2014-06-29 09:41:16	0101000000000000
225	2014-06-29 09:41:33	0101000000000000
226	2014-06-29 09:41:49	0101000000000000
227	2014-06-29 09:42:07	0101000000000000
228	2014-06-29 09:42:24	0101000000000000
229	2014-06-29 09:42:39	0101000000000000
230	2014-06-29 09:42:57	0101000000000000
231	2014-06-29 09:43:14	0101000000000000

Slika 11: Tablica sa zabilježenim položajima objekta

Tablica je kreirana pomoću jednostavne sintakse:

```
CREATE TABLE points (
id serial NOT NULL,
the_time time
geom geometry (Point)
);
```

3.2.3. Vizualizacija kretanja objekta

Vizualizacija kretanja objekta u realnom vremenu postignuta je korištenjem HTML-a, CSS-a i javascripta. Navedene tehnologije često se koriste zajedno za izradu interaktivnih web gis aplikacija.

Karta je smještena u jedinstveni prostor - *map*. Koristeći CSS, definirao sam stil tog prostora na način da karta bude na cijelom ekranu. Glavni je sloj karta koju sam dizanirao koristeći Mapbox. Ona je ubačena u kartu pomoću *leaflet.js* javascript biblioteke. Dodatno sam radi lakšeg testiranja i praćenja definirao inicijalan položaj i razinu uvećanja karte.

Kako bi se zadovoljili zahtjevi sustava za praćenje, potrebno je redovito ažurirati položaj objekta. U tu svrha kreirana je funkcija *update_position()*. Koristio sam jQueryjevu metode *getJSON()* koja omogućuje učitavanje JSON strukture podataka sa servera koristeći GET HTTP zahtjev. GET je često korišten zahtjev koji nalaže serveru da povuče podatke koji su spremljeni na stranici. *GetJSON* metoda je strukturiran na sljedeći način (url 18):

$.getJSON(url [, data] [, success])

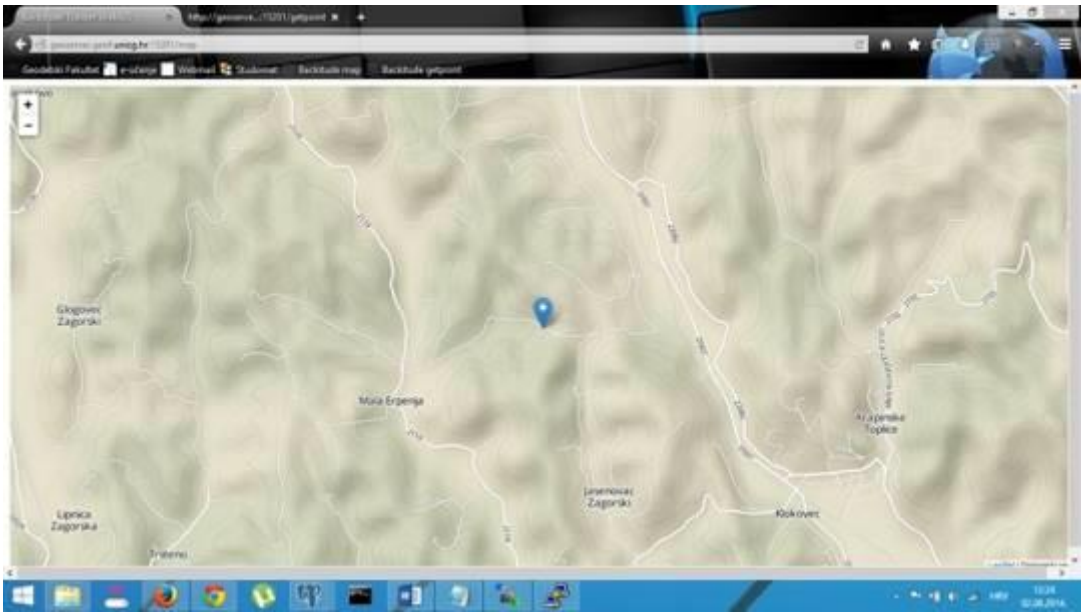
Prvi parametar metode je url tipa string koji sadrži url stranice na koji je poslan zahtjev. Drugi parametar je *data* tipa *Plain Object*. *Plain Object* je javascript objekt koji sadrži nula ili više *key:value* parova. *Success* je funkcija koja se izvršava ako je zahtjev uspješan.
U mojem slučaju metoda *getJSON()* šalje zahtjev serveru da povuče podatke sa navedenog url-a koji vodi na stranicu koja je definirana u btwebgis.py aplikaciji. Stranica */getpoint* sadrži samo najažurniji položaj objekta u geoJSON strukturi. Ako je zahtjev uspješan, izvršava se funkcija s parametrom *data* koji sadrži sve podatke povučene sa stranice. Poznavajući strukturu geoJSON podataka, lako je pristupiti konkretnoj informaciji unutar geoJSON formata. U ovom su slučaju to geografska širina i duljina. Obje se nalaze u dijelu geoJSON-a s koordinatama, a koordinate su unutar dijela s geometrijom. U istom potezu odlučio sam smjestiti sredinu prikaza karte na novoodređeni položaj objekta. Položaj objekta prikazan je

novi objektom, tj. markerom koji prima vrijednosti geografske širine i duljine te se smješta na kartu. Njegovo ažuriranje izvršava se ponavljanim pozivanjem funkcije *update_position*.

Kada bi samo jednom pozvali funkciju, bilo bi potrebno za svaki novi položaj ručno osvježiti stranicu. Stoga je potrebno napraviti vremensku petlju. Jedan od načina je pozvati metodu *setInterval()*. Ona se često zna zamijeniti sa *setTimeout()* metodom. Razlika između njih je ta što prva čeka određeni broj milisekundi, izvrši se te se nastavlja izvršavati svakih unaprijed određenih milisekundi. Metoda se može zaustaviti pomoću *clearInterval()* metode koja se može aktivirati primjerice klikom miša na gumb koji aktivira tu metodu. Za razliku od prve, *setTimeout()* metoda se izvršava samo jednom, i to nakon određenog broja milisekundi (URL 19). Za ovaj slučaj korištena je *setInterval()* metoda.
Prvi parametar metode je funkcija koja će se izvršavati svakih n milisekundi koje definiramo u drugom parametru metode.

```
setInterval(update_position, 2000);
update_position();
```

Na slici 12 prikazan je izgled prototipa za praćenje objekta.



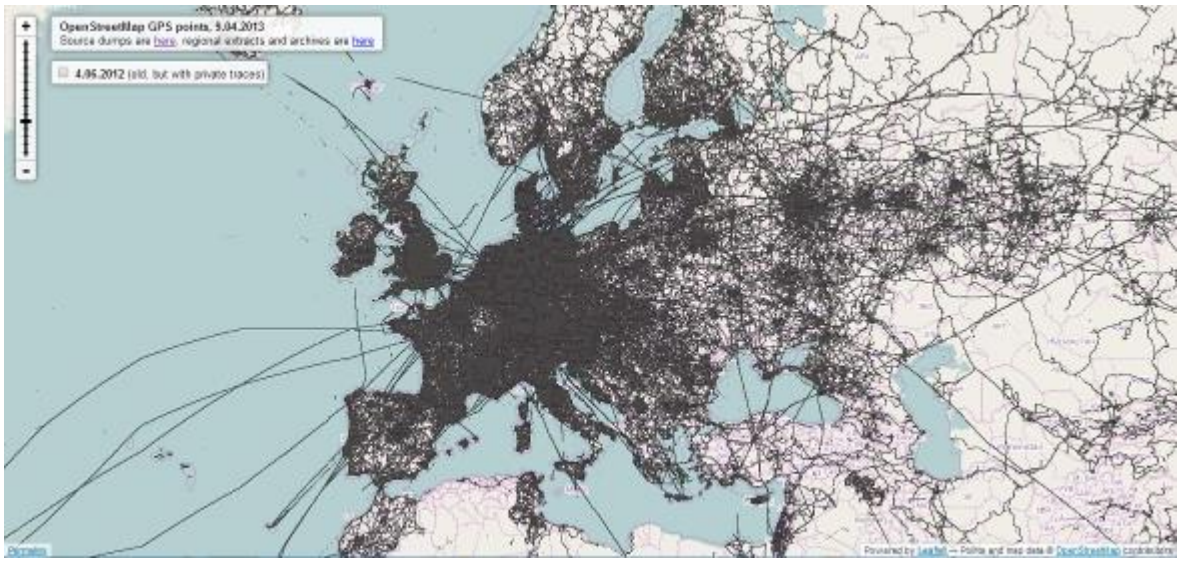
Slika 12: Izgled prototipa za praćenje objekata u realnom vremenu

3.3. Ispitivanje mogućnosti Postgisa

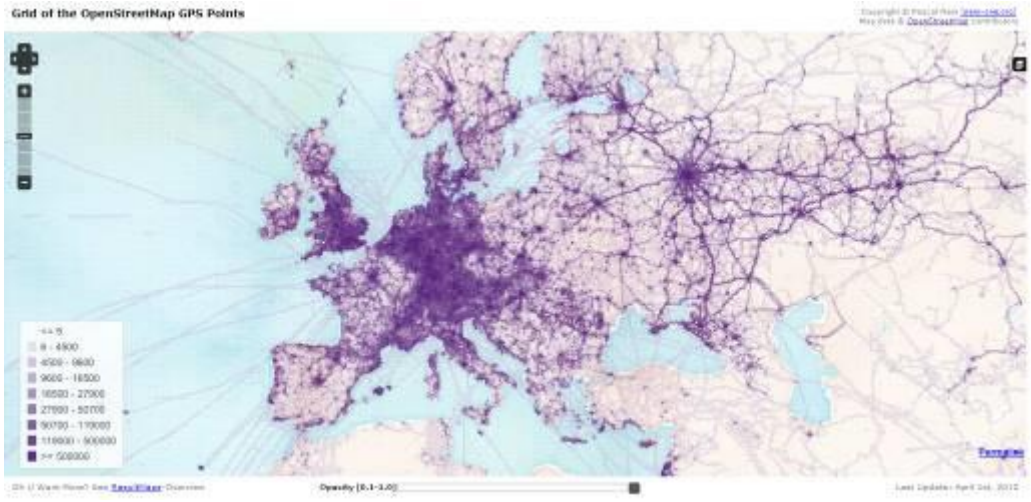
3.3.1. Izvor

Podaci koji će biti korišteni u testiranju skinuti su sa Openstreetmap wiki stranice. Postoji mogućnosti skidanja podatka za cijeli svijet, zatim po kontinentima te po državama. Njihova arhiva sadrži kolekciju GPS točaka iz prvih sedam i pol godina rada OpenStreetMap servisa. Radi se o sirovim podacima, odnosno GPS logova koje su korisnici opažali sa svojim uređajima te pohranili na server. Najčešće se radi o logovima koji su zabilježeni svake jedne sekunde ili svakog jednog prijeđenog metra. Rezultat jedne takve „izmjere“ je serija točaka koja prikazuje putanju korisnika. Ti podaci mogu biti korišteni za dodavanje novih objekata na kartu, slično kao što je to moguće sa zračnim snimkama (url 20).

Za ispitivanje baze odlučio sam koristiti GPS logove sa područja Njemačke. Lokacija podataka nije bitna, ali je bitna količina podataka. Količina podataka za ovo istraživanje ograničena je memorijom računala. Na slici 13 prikazana je karta sa svim GPS logovima na području Europe, a na slici 14 karta gustoće.



Slika 13: Karta GPS logova na području Europe



Slika 14: Karta gustoće

3.3.2. Struktura podataka

Skinuta arhiva za određeno područje sastoji se od tri mape: *identifiable*, *public* i *trackable*. Public mapa sadrži gpx datoteke koje nemaju podatke o vremenu, ali će se također koristiti u testiranju jer vrijeme nije neophodan podatak za ovaj slučaj. Gpx datoteke nalaze se u podmapama i svaka od njih ima jedinstven naziv.

Svaka gpx datoteka unutar tih mapa strukturirana je na idući način:

```
<?xml> definiranje xml datoteke i verzije
<gpx> početak gpx datoteke
    <trk> pojedina putanja
        <name> ime putanje
        <number> broj putanje
        <trkseg> segment putanje
            <trkpt> točka unutar segmenta putanje
                lat geografska širina
                lon geografska duljina
                <ele> visina </ele>
                <time> datum i vrijeme </time> # ne postoji u public mapi
            </trkpt>
        </trkseg>
    </trk>
</gpx>
```

Naziv datoteke, ime, broj i segmenti putanje zajedno sa geografskom širinom i duljinom bit će pohranjeni u bazu podataka.

Svaka datoteka sadrži više putanja. Svaka putanja sastoji se od više segmenata, a svaki segment od više točaka.

Primjer jedne gpx datoteke:

```
<?xml version='1.0' encoding='utf-8'?>
<gpx xmlns="http://www.topografix.com/GPX/1/0" version="1.0" creator="OSM gpx_dump.py">
  <trk>
    <name>Track 0</name>
    <number>0</number>
    <trkseg>
      <trkpt lat="43.7432467" lon="15.7864933">
        <ele>-17.40</ele>
        <time>2008-09-02T12:48:55Z</time>
      </trkpt>
      <trkpt lat="43.7433017" lon="15.7863867">
        <ele>-14.60</ele>
        <time>2008-09-02T12:49:40Z</time>
      </trkpt>
    </trkseg>
  </trk>
</gpx>
```

3.3.3. Obrada podataka

Podaci se na različiti način mogu ubaciti u bazu podataka. Prvi pokušaj bio je direktno ubacivanje u bazu koristeći *beautiful soup* parser i *insert* naredbe. Beautiful soup je dodatak za Python koji omogućuje čitanje i manipulaciju xml datoteka. Međutim, u kombinaciji sa insert naredbom nije zadovoljio u pogledu brzine unosa. Ubacivanje bi trajalo predugo pa je korišten drugi način.

Alternativa insert-u je *copy* naredba. Ona omogućuje brzo kopiranje podataka između PostgreSQL-a i standardnih datoteka, npr. csv datoteke. Pošto su skinuti podaci u gpx datoteci, potrebno je izvući točke i pohraniti ih u csv datoteku. U tu svrhu napisana je Python skripta za unos točaka iz svih gpx datoteka u jednu csv datoteku.

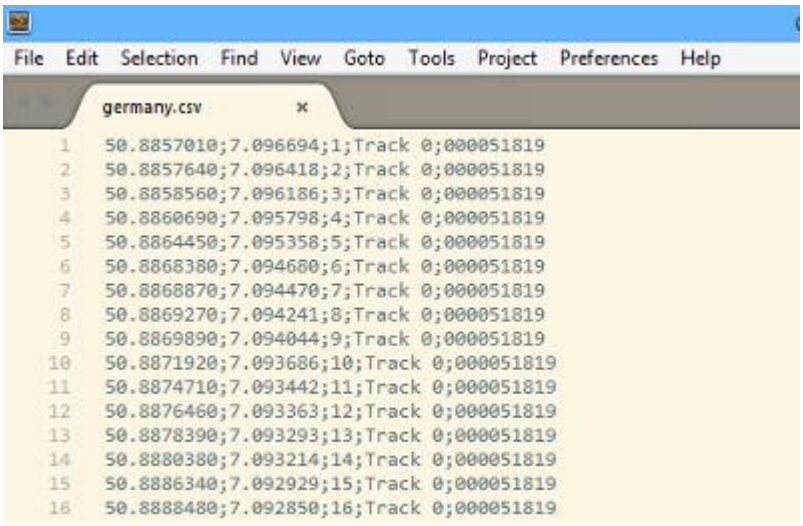
3.3.3.1 Izrada skripte za unos u csv datoteku

Za čitanje svake gpx datoteke unutar svake podmape zahtijeva poseban dio koda koji prolazi kroz sve podmape u odnosu na početni direktorij i sprema sve nazive datoteka u listu.

U idućem koraku prolazi se kroz svaku gpx datoteku unutar te liste. Otvara se jedna po jedna datoteka. Varijabla *current_file* sprema naziv datoteke koji je jedinstven. Unutar te petlja nalazi se jedna petlja koja prolazi kroz redove unutar gpx datoteke i traži oznaku *„<name>“* koja sadrži naziv putanje. Naziv putanje pohranjuje se u varijablu *current_track*. Kako bi se sačuvao poredak točaka unutar putanje, dodan je brojač *seq* koji se povećava za jedan nakon što se pronađu sve točke unutar putanje, a poništava se na nulu kad počinje druga putanja.

Geografska širina (*lat*), duljina (*lon*), naziv putanje (*current_track*) i naziv datoteke (*current_file*) čine jedan zapis (jedan položaj točke) te se ispisuje u jedan red u csv datoteku odvojeni sa ";".

Na slici 15 prikazan je dio datoteke unutar koje su pohranjene sve točke. Prebačeno je ukupno 442955924 točaka što na disku zauzima 18 GB.



Slika 15: Dio csv datoteke u koju su pohranjene točke

3.3.4. Unos u bazu podataka

Za unos u bazu podataka korištena je sql-ova naredba *copy from*. Korišten je idući sql kod:

```
COPY raw_data(lat, lon, seq, track, filename)
FROM 'C:\\Python27\\germany.csv'
WITH DELIMITER ';';
```

Kod unosa potrebno je definirati izvornu datoteku (*germany.csv*), tablicu u koju unosimo (*raw_data*) te u koje stupce će ubaciti određene podatke (*lat, lon, seq, track, filename*). Ti podaci odvojeni su sa „;“. Tablica *raw_data* je neprostorna tablica koja ne sadrži geometriju, već samo vrijednosti geografskih širina i duljina kao brojčani tip. Ona će u daljnjem postupku biti osnova iz koje će se kreirati geometrijski zapis točaka, a zbog dodatnih stupaca (*seq,track,filename*) moguće je rekonstruirati originalne GPS putanje.

Copy naredba optimizirana je za unos velikih količina podataka. Nema toliko mogućnosti kao insert, ali zato prilikom unosa velike količine podataka nastaje manje opterećenja. Zbog toga je kod takvih unosa *copy* gotovo uvijek brža metoda. (url 21).

3.3.5. Testiranje

U poglavlju koje slijedi opisani su odabrani testovi i protokol njihova izvršavanja, kao i računalo i softver na koji su korišteni za testiranje.

3.3.5.1 Računalo i softver za testiranje

Kod testiranja korišteno je prijenosno računalo Toshiba Satellite sa specifikacijama u tablici 1.

Model	L850-1EZ
OS	Windows 8.1 Pro (64bit)
Procesor	Intel Core i5-3210M (4 jezgre, 2.5 GHz)
RAM	4 GB DDR3
Hard disk	SATA 640 GB (5400 rpm)
L2 Cache	2x256 KB
L3 Cache	3 MB

Tablica 1: Specifikacije računala

Testiranje je izvršeno unosom SQL naredbi u SQL Query prozor pgAdmin III softvera. Pritom je korišten PostgreSQL verzija 9.3. i PostGIS verzija 2.1.1.

3.3.5.2 Odabrani testovi

Testiranje je podijeljeno na dva dijela: Testiranje točaka i testiranje linija nastalih iz istih točaka.

U prvom dijelu, podaci iz tablice *raw_data* koristit će se za popunjavanje novih tablica. Nove tablice su iduće:

1. *Points_geog* definira točku kao geometriju na WGS84 elipsoidu
2. *Points_geom* definira točku kao geografiju na WGS84 elipsoidu
3. *Points_proj* definira točku kao geometriju, ali u projekciji. Odabrana je „Pseudo Mercator“ projekcija (EPSG:3857) jer je ona jedinstvena za cijeli svijet.

Tablice točaka kreirane su idućim SQL naredbama:

```
CREATE TABLE points_geog (
gid serial NOT NULL,
geog geography(Point,4326),
seq integer,
track text,
filename text
) tablespace postgis_benchmark;

CREATE TABLE points_geom (
gid serial NOT NULL,
geom geometry(Point,4326),
seq integer,
track text,
filename text
) tablespace postgis_benchmark;

CREATE TABLE points_proj (
gid serial NOT NULL,
geom geometry(Point,3857),
seq integer,
track text,
filename text
) tablespace postgis_benchmark;
```

Važna stavka testiranja jest obaveza da u svakoj tablici budu isti podaci. Isprobana su dva načina. Prvi način bio je popunjavanje svake tablice iz izvorne *raw_data* tablice. Da bi podaci bili isti, potrebno ih je poredati kod unosa prilikom korištenja *limit* opcije. Navedena opcija se koristi u kombinaciji sa *offsetom* za postupno popunjavanje tablica. Međutim, prvi je način spor zbog čestih i dugotrajnih stvaranja poredka. Zbog toga je korišten drugi i brži način. Tablica *Points_geog* popunjena je podacima iz tablice *raw_data* iz kreiranje geografskog tipa. Tablica *Points_geom* popunjena je podacima iz tablice *Points_geog*, što je pojednostavljeno odabirom istog elipsoida (nije potrebna transformacija). Tablica *Points_proj* popunjena je podacima iz *Points_geom* tablice uz projekciju točaka.

Broj zapisa u tablicama mijenjao se istim redom na načina da su prvo u tablicu *Points_geog* ubačene 4 točke, koje su zatim prebačene u *Points_geom* te iz *Points_geom* u *Points_proj*. Idući korak je povećanje prve tablice na 40 zapisa istim postupkom druge i treće tablice. Broj točaka se povećavao za 10 puta, sve do maksimalnog broja točaka u pojedinoj tablici (400 milijuna) zbog ograničenja računala.

U međukoracima se kreiraju i brišu indeksi te se prati veličina tablice, indeksa i toast tablica, kao i vrijeme unosa jedne točke. Iz tablice *Points_geog* za svaki odabrani broj točaka osim najvećeg (4,40,400...,...40 milijuna) spremljene su linije kreirane iz tih točaka u csv datoteku. Pritom je ukupno kreirano 8 csv datoteka koje će se koristiti u drugom dijelu testiranja.

Cilj je spoznati omjer veličine tablice, pripadajućeg indeksa i toast tablice za određeni broj točaka, kao i odnos veličina tablica, indeksa i toast tablica različitih načina zapisa točaka. Važnu komponentu testiranja predstavlja vrijeme unosa jedne točke u tablicu sa 4 postojeća zapisa, zatim 40 i tako redom do unosa jedne točke u tablicu sa 400 milijuna zapisa. Cilj je spoznati kako veličina tablice utječe na unos točke, odnosno insert naredbu.

Nakon testiranja unosa točaka, pristupa se testiranju linija. Tablice s linijama kreirane su na sličan način kao i toče. Linije s geografskim tipom kreirane su iz csv datoteka, geometrijske iz geografskih te one u projekciji transformacijom iz geometrijskih. Za svaki način definiranja linija (geometrija, geografija, geometrija s

projekcijom) i za svaki određeni broj točaka osim najvećeg (4,40,400...,...40 milijuna) kreirano je ukupno 24 novih tablica te su popunjene linijama istim linijama.

```
CREATE TABLE lines_geom_[broj_točaka] (
gid serial NOT NULL,
geom geometry(LineString,4326),
track text,
filename text
) tablespace postgis_benchmark;

CREATE TABLE lines_geog_[broj_točaka] (
gid serial NOT NULL,
geog geography(LineString,4326),
track text,
filename text
) tablespace postgis_benchmark;


CREATE TABLE lines_proj_[broj_točaka] (
gid serial NOT NULL,
geom geometry(LineString,3857),
track text,
filename text
) tablespace postgis_benchmark;
```

Uspoređuje se veličina tablica, indeksa i toast tablica ovisno o broju točaka iz kojih su kreirane linije. Budući da kod linija nema testiranja vremena unosa točke, isključuje se tablica sa 400 mil. zapisa. Dodatni test čini usporedba veličina putanje zapisane u točkama i u linijama.

Dodatni test bio je usporedba veličina tablica, toast tablica i indeksa između linija kreiranih iz istih točaka, ali različitih broja linija. Naime, povezivanjem svih putanja unutar iz jedne gpx datoteke u jednu putanju, dobiva se manji broj linija te nas zanima kako to utječe na već spomenute veličine.

3.3.5.3 Protokol testiranja

Protokol testiranja sadrži opis testiranja korak po korak. Sastoji se od osam koraka, sukladno odabranim testiranjima.

1. Unos 4 točke u tablicu *points_geog* iz tablice *raw_data*. Kod unosa kreira se geometrija točke iz geografske duljine (lon) i geografske širine (lat) te se pridružuju atributi *seq*, *track* i *filename*.

```
INSERT INTO points_geog (geog, seq, track, filename) SELECT
(ST_SetSRID(ST_MakePoint(lon, lat), 4326)),seq, track, filename FROM raw_data limit 4
offset 0;
```

2. Unos 4 točke u tablicu *points_geom* iz tablice *points_geog.*

```
INSERT INTO points_geom (geom, seq, track, filename) SELECT  geog::geometry,seq,
track, filename FROM points_geog limit 4 offset 0;
```

3. Unos 4 točke u tablicu points_proj iz tablice points_geom. Kod unosa se izvršava transformacija, tj projekcija u novi referentni sustav.

```
INSERT INTO points_proj (geom, seq, track, filename) SELECT
ST_Transform(geom,3857),seq, track, filename FROM points_geom limit 4 offset 0;
```

4. Kreiranje indeksa na geometrijskim, odnosno geografskim poljima koristeći GiST. Veličine indeksa, tablica i toast tablica zapisuju se u excel tablicu.

```
CREATE INDEX pt_geog on points_geog USING gist (geog);
CREATE INDEX pt_geom on points_geom USING gist (geom);
CREATE INDEX pt_proj on points_proj USING gist (geom);
```

5. Brisanje indeksa da se ne usporava vrijeme unosa idućeg broja točaka.

```
DROP INDEX pt_geog;
DROP INDEX pt_geom;
DROP INDEX pt_proj;
```

6. Spremanje linija kreiranih iz 4 točke u csv datoteku, odvojeno za svaki broj zapisa.

COPY (SELECT
ST_MakeLine(geog::geometry ORDER BY seq ASC)::geography, track, filename
FROM points_geog
GROUP BY filename, track) TO 'E:\\postgres\\linestrings\\lines_geog_[broj_zapisa].csv'
WITH DELIMITER ';';

7. Ponavljanje prvih 7 koraka za 40, 400, 4 000, 40 000, 400 000, 4 000 000 i 40 000 000 zapisa. Pritom se mijenjaju vrijednosti *limit/offseta* kod unosa te nazivi datoteka i tablica kod kreiranja i unosa linija.

8. Kreiranje linija iz csv datoteka.

COPY lines_geog_[broj_zapisa](geom, track, filename)
FROM 'E:\\postgres\\linestrings\\lines_geog_[broj_zapisa]..csv'
WITH DELIMITER ';';

INSERT INTO lines_geom_[broj_zapisa] (geom, track, filename) SELECT geog::geometry, track, filename FROM lines_geog_[broj_zapisa];

INSERT INTO lines_proj_[broj_zapisa] (geom, track, filename) SELECT ST_Transform(geom,3857),track, filename FROM lines_geom_[broj_zapisa];

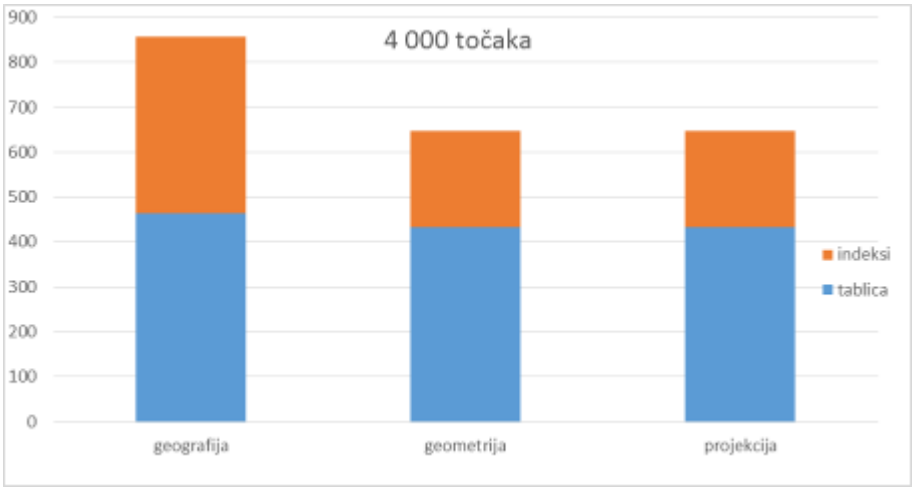
9. Kreiranje indeksa na geometrijskim, odnosno geografskim poljima u tablicama s linijama koristeći GiST. Veličine indeksa, tablica i toast tablica zapisuju se u excel tablicu.

CREATE INDEX pt_geog on lines_geog_[broj_zapisa] USING gist (geog);
CREATE INDEX pt_geom on lines_geom_[broj_zapisa] USING gist (geom);
CREATE INDEX pt_proj on lines_proj_[broj_zapisa]USING gist (geom);

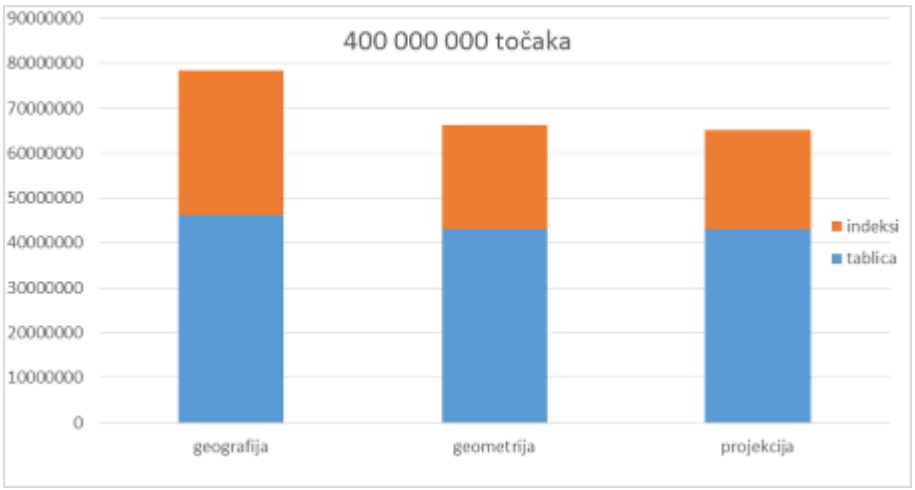
4. Rezultati

Test 1: Usporedba veličina indeksa i tablica ovisno o tome radi li se o geometriji, geografiji ili geometriji u projekciji.

Tablice i indeksi geometrije na elipsoidu i u projekciji imaju slične vrijednosti , dok za geografski tip imaju malo veće vrijednosti za tablicu, a osjetno veće za indekse. Rezultat je sličan za 4 000 (graf 1) i za 400 000 000 točaka (graf 2) .



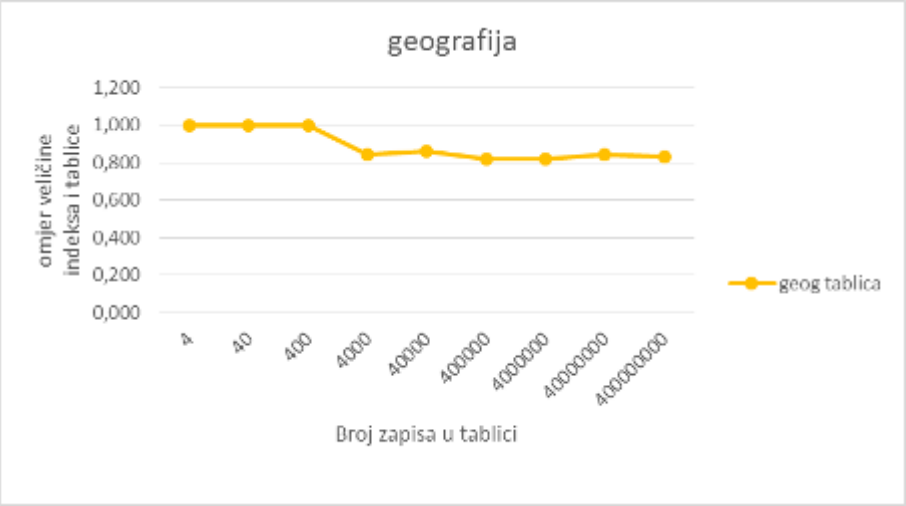
Graf 1: Rezultat za 4 000 točaka



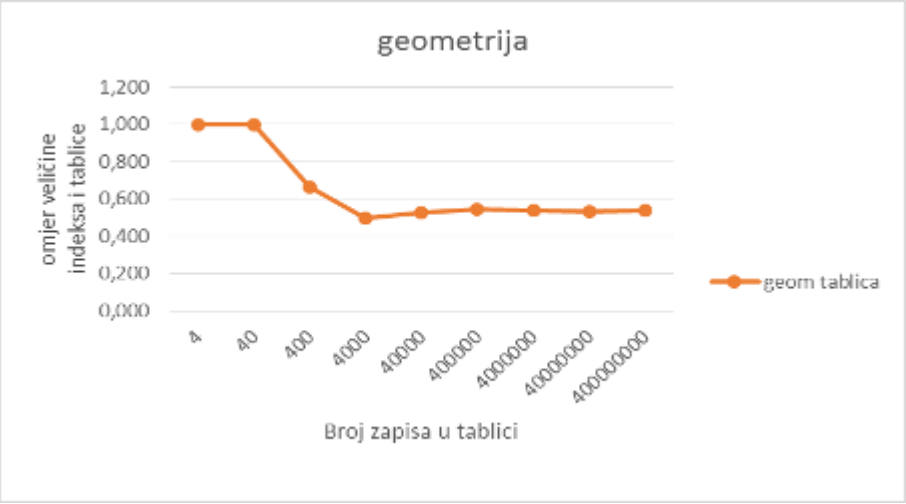
Graf 2: Rezultat za 400 000 000 točaka

Test 2: Omjer veličina indeksa i tablica u ovisnosti o broju točaka za geografske, geometrijske i podatke u projekciji.

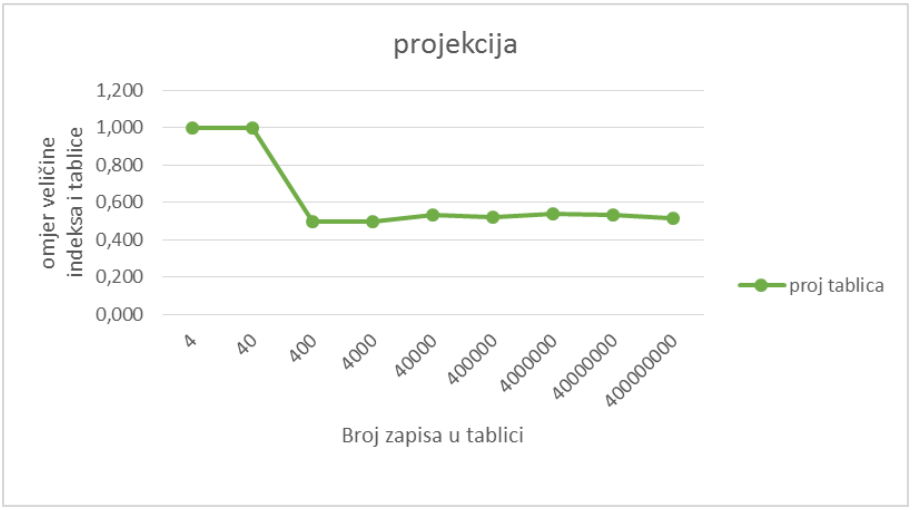
Rezultati na grafovima 3 i 4 prikazuju sličan omjer za geometriju na elipsoidu i u projekciji te on iznosi malo više od 0.5. Omjer za geografiju na grafu 5 je veći te on iznosi više od 0.8.



Graf 3: Omjer veličine indeksa i tablice za geografski tip

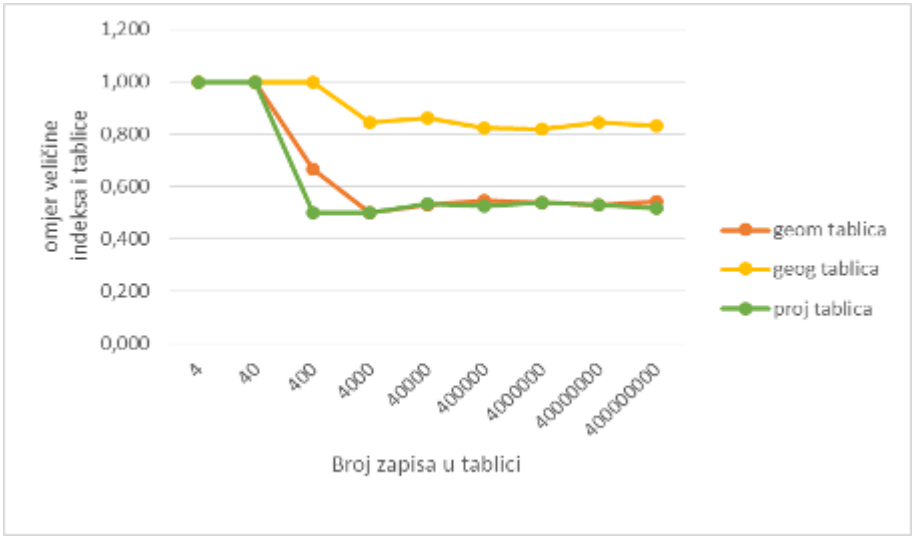


Graf 4: Omjer veličine indeksa i tablice za geografski tip



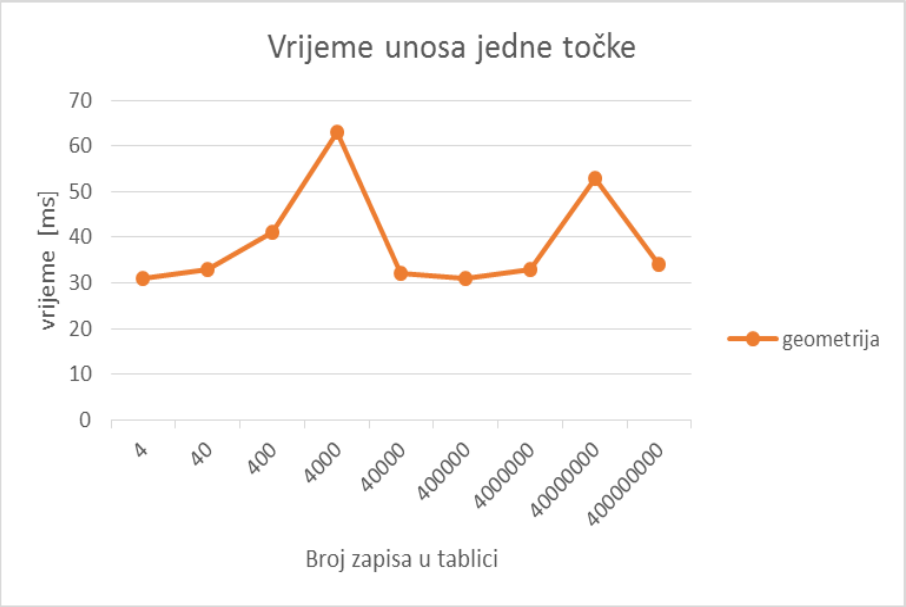
Graf 5: Omjer veličine indeksa i tablice za geometriju u projekciji

Graf 6 prikazuje ukupan prikaz omjera za sve načine zapisivanja prostornih podataka.



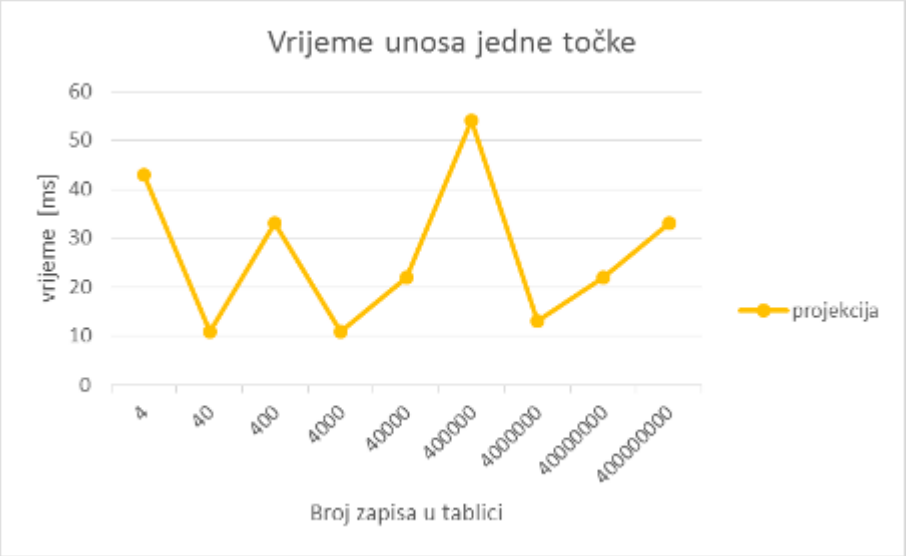
Graf 6: Ukupan prikaz omjera

Test 3: Vrijeme unosa jedne točke u tablice geometrije (graf 7), geografije (graf 8) i geometrije u projekciji (graf 9) u ovisnosti o broju točaka koje su već pohranjene u tablici.

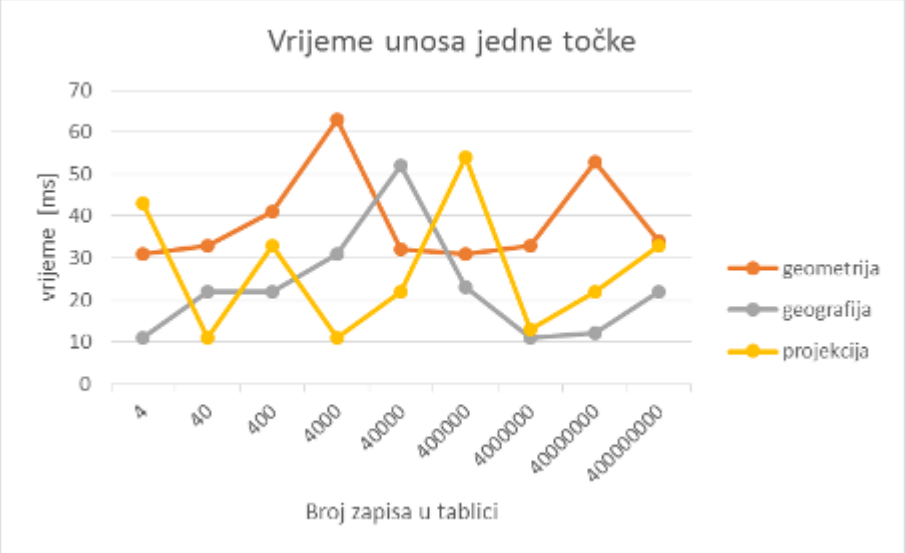


Graf 7: Vrijeme unosa u ovisnosti o broju točaka za geometriju

Graf 8: Vrijeme unosa u ovisnosti o broju točaka za geografiju



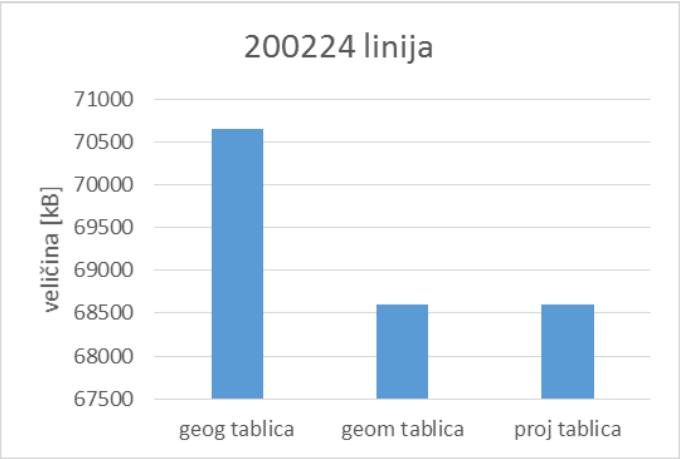
Graf 9: Vrijeme unosa u ovisnosti o broju točaka za projekciju



Graf 10: Ukupan prikaz vremena unosa

Test 4: Usporedba veličina tablica, toast tablica i indeksa za geografiju, geometriju i projekciju. Prikazani su samo rezultati za 200224 linija koje su nastale iz 40 milijuna točaka.

Tablica sa geografskim tipom je najveća, a tablice za geometriju imaju jednake vrijednosti (graf 11).

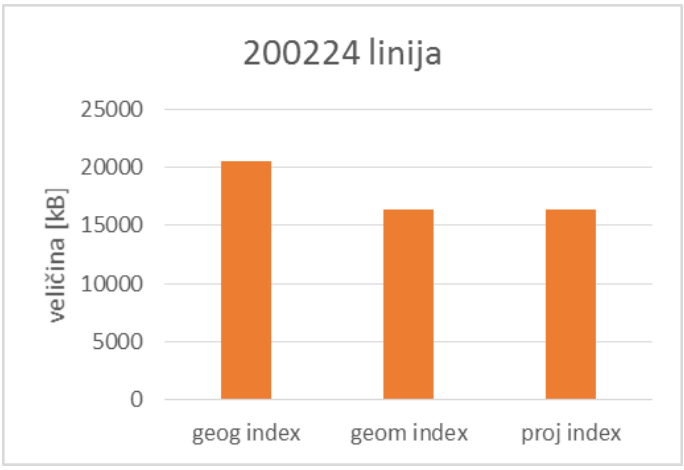


Graf 11: Veličina tablica

Veličina toast tablice najveća je za projekciju, dok ona za geografiju ima najmanju vrijednost (graf 12).

Graf 12: Veličina toast tablica

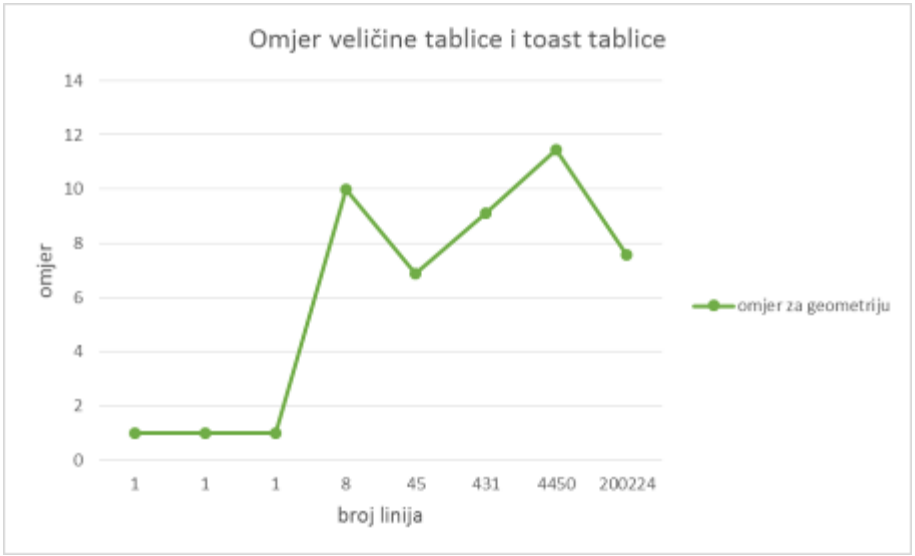
Veličina indeksa najveća je za geografiju, dok su veličine za projekciju geometriju jednake (graf 13).



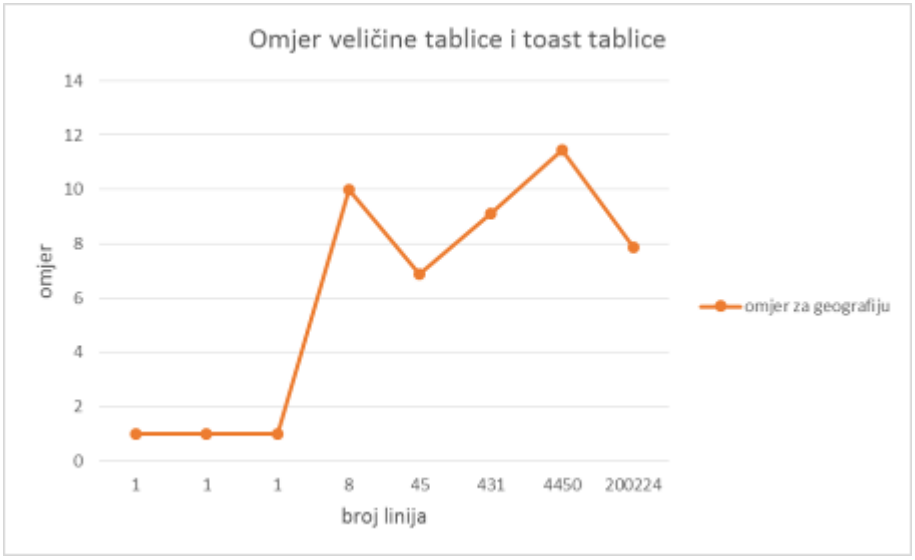
Graf 13: Veličina indeksa

Test 5: Omjer veličine tablice i toast tablice za geometriju, geografiju i projekciju ovisno o broju linija.

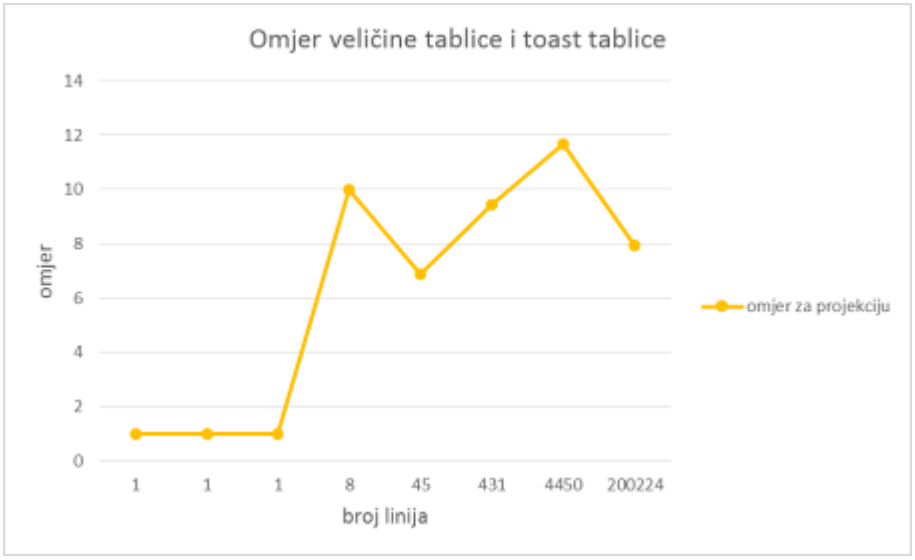
Kod geometrijskog tipa se vrijednost omjera kreće oko 9 osim za jednu liniju (graf 14). Geografski tip i projekcija imaju sličan trend što je vidljivo na grafovima 15 i 16.



Graf 14: Omjer veličina tablica i toast tablica za geometriju



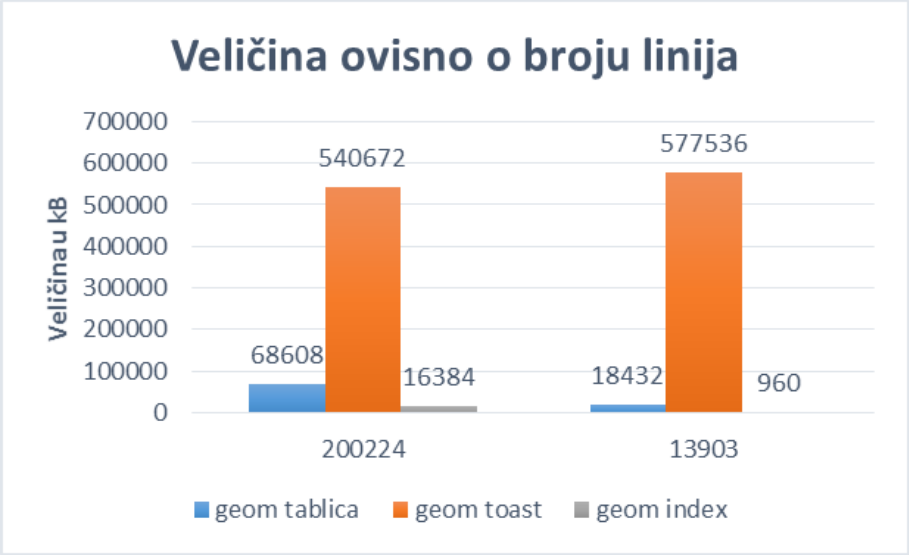
Graf 15: Omjer veličina tablica i toast tablica za geografiju



Graf 16: Omjer veličina tablica i toast tablica za projekciju

Test 6: Usporedba veličina tablica, toast tablica i indeksa ovisno o broju linija koje su nastale iz istih točaka u geometrijskom zapisu.

Kod većeg broja nastalih linija veće su i veličine tablica i indeksa, ali je veličina toast tablica manja (graf 17).



Graf 17: Usporedba veličina ovisno o broju linija

Grafovi iz rezultata izrađeni su na temelju vrijednosti iz slijedećih tablica dobivenih vlastitim testiranjima.

broj zapisa	geog tablica	geom tablica	proj tablica	geog index	geom index	proj index
4	8	8	8	8	8	8
40	8	8	8	8	8	8
400	48	48	48	48	32	24
4000	464	432	432	392	216	216
40000	4576	4272	4272	3944	2264	2288
400000	46080	43008	43008	37888	23552	22528
4000000	456704	427008	427008	374784	229376	229376
40000000	4571136	4267008	4267008	3861504	2264064	2262016
400000000	46065280	42991616	42991616	32234464	23236568	22197192

Tablica 2: Veličine tablica i indeksa za točke

Broj zapisa	geometrija	geografija	projekcija
4	1,000	1,000	1,000
40	1,000	1,000	1,000
400	0,667	1,000	0,500
4000	0,500	0,845	0,500
40000	0,530	0,862	0,536
400000	0,548	0,822	0,524
4000000	0,537	0,821	0,537
40000000	0,531	0,845	0,530
400000000	0,540	0,834	0,516

Tablica 3: Omjer veličina tablica i indeksa

br zapisa	geometrija	geografija	projekcija
4	31	11	43
40	33	22	11
400	41	22	33
4000	63	31	11
40000	32	52	22
400000	31	23	54
4000000	33	11	13
40000000	53	12	22
400000000	34	22	33

Tablica 4: Vrijeme unosa jedne točke

br točaka	br linija	geog tablica	geom tablica	proj tablica	geog toast	geom toast	proj toast	geog index	geom index	proj index
4	1	8	8	8	8	8	8	8	8	8
40	1	8	8	8	8	8	8	8	8	8
400	1	8	8	8	8	8	8	8	8	8
4000	8	8	8	8	80	80	80	8	8	8
40000	45	80	80	80	552	552	552	8	8	8
400000	431	600	600	584	5472	5472	5520	40	32	40
4000000	4450	4920	4912	4912	56320	56320	57344	472	456	456
40000000	200224	70656	68608	68608	536576	540672	545792	20480	16384	16384
40000000	13903		18432			577536			960	

Tablica 5: Veličine tablica, toast tablica i indeksa za linije

5. Diskusija

U prvom testu pokazalo se da geografija ima najveću veličinu tablica, kao i veličinu indeksa. Očekivano, geometrija i projekcija su imale slične rezultate u ovom testu, dok je do malih varijacija došlo zbog različitih referentnih sustava. Povećanjem broja točaka ostali su slični relativni odnosi, pa su u rezultatima prikazani samo grafovi za 4 tisuće i 400 milijuna točaka. Veličina toast tablice kod točaka nije imala veću ulogu. To je i očekivano pošto jedna točka nema veliki zapis pa stane na jednu stranicu koja zauzima svega 8 kilobajta. Kod manjih unosa podataka, primjerice za četiri ili četrdeset točaka, veličine tablica i indeksa također su zauzimale 8 kilobajta. To je minimalna veličina stranice koja se može povećati, ali kod točkastih zapisa ne bi bilo ekonomično. Već kod navedenog manjeg broja zapisa stranice su imale slobodnog neiskorištenog prostora.

U drugom testu pratio se međusobni omjer veličine tablica i indeksa. Veličina indeksa kod geometrije na elipsoidu i projekcije varirala je oko 50% veličine tablice, dok se kod geografije ta vrijednost kretala oko 80% veličine tablice. Odstupanja su se pojavila na malim količinama podataka, što je očekivano jer se radi o minimalnim veličinama koje su iste za indeks i za tablicu.

U trećem testu pratilo se vrijeme unosa jedne točke u tablicu sa različitim brojem zapisa. Iako sam očekivao da će se vrijeme unosa povećavati kod velikog broja zapisa, to se nije dogodilo. Bez obzira na veličinu tablice ili način zapisivanja prostornih podataka, vrijeme unosa bilo je uvijek između desetak i šezdesetak milisekundi. Za varijacije u ovom slučaju možemo reći da su slučajnog karaktera, jer ne ovise u veličini tablice, već o trenutnim performansama softvera i računala. Moja pretpostavka je da je za povećanje vremena potrebna ili puno veća količina točaka u tablici zajedno sa indeksom, ili pohraniti više točaka kao linije što bi dovelo do preslagivanja indeksa i povećavanja vremena unosa. Međutim, u tom slučaju preporuča se brisanje indeksa prije unosa i njegovo ponovno kreiranje što bi skratilo vrijeme unosa vremena.

Četvrti test pratio je veličine tablica, indeksa i toast tablica koje su pokazale zanimljive rezultate. Očekivano, tablice s linijama su zauzimale manje mjesta, a kao i kod točaka, najviše je mjesta zauzeo geografski tip. Geometrija na elipsoidu u projekciji imaju slične veličine tablica. Međutim, toast tablice su najveće kod geometrije u projekciji. To je posljedica većeg broja znamenki jedne koordinate za projekciju u odnosu na elipsoidne koordinate. Kod točaka u projekciji to nije bio slučaj jer je zapis jedne točke kratak i stane na jednu stranicu. Indeksi nisu pokazivali veća odstupanja u odnosu na točke, te su relativni odnosi sačuvani, tj. geografski tip ima i dalje najveći indeks, dok geometrija u projekcija imaju slične vrijednosti .

Peti test pratio je omjer veličina toast tablica i tablica kod različitih načina zapisa. Omjer je sličan za geografiju, geometriju na elipsoidu i geometriju u projekciji. Najveći omjer bio je očekivano za projekciju (prosjek 9,20), ali nije puno odstupao od omjera za geografiju (prosjek 9,07), odnosno od omjera za geometriju na elipsoidu (prosjek 9,01).

Šesti test prikazuje da se većim brojem linija kreiranih iz istog broja točaka povećava veličina tablice i indeksa, ali je veličina toast tablice manja. Rezultat je očekivan jer se dio podataka zaglavlja ponavlja što povećava tablicu. Manjim brojem kreiranih linija iz istog broja točaka linije su duže, imaju više točaka koje ne stanu u jednu stranicu pa se gomilaju i kompresiraju u toast tablice.

6. Zaključak

Sustavi za praćenje važan su faktor u svijetu u kojem je potreba za praćenjem objekata sve veća. Za izradu sustava za praćenje potrebno je pažljivo odabrati tehnologije koje će se koristiti. Svaka od njih ima svoje prednosti i nedostatke koji mogu utjecati na performanse sustava. Izradom vlastitog prototipa za praćenje objekata naučio sam što se nalazi u pozadini jednog takvog sustava. Iza jedne točke na ekranu koja predstavlja položaj objekta krije se niz koraka i tehnologija. Težište rada bilo je na bazama prostornih podataka koje pohranjuju položaje objekata. Testiranja su se odnosila na učinkovitost PostgreSQL/PostGIS baze podataka za primjenu u sustavu za praćenje.

Prototip sustava za praćenje zapisivao je točke sa geometrijom na elipsoidu. Rezultatima istraživanja došao sam do zaključka kako je to dobar, ali ne i idealan način praćenja korisnika. Odabir geometrije na elipsoidu kao način pohrane prostornih podataka pokazao se dobrim jer njegova tablica, kao i indeks, zauzimaju manje prostora od geografskog tipa. Geometrija u projekciji davala je slične rezultate kao i geometrija na elipsoidu, ali je svoje mane pokazala u linijskom zapisu gdje je veličina njezine toast tablice bila najveća. Omjer indeksa i tablica s točkama ne mijenja se puno povećavanjem tablica, ali je on kod goegrafskom tipa najveći. Zanimljive rezultate dala su ispitivanja linija. Spremanjem podataka praćenja objekta u linije može se smanjiti veličina tablice, kao i indeksa. Međutim, zbog gustoće točaka gomilaju se linije s puno čvorova koje ne stanu u jednu fiksnu stranicu pa se reduciraju i spremaju u zasebne toast tablice. One su za zadan skup podataka bile oko 9 puta veće od samih tablica. Problem bi se mogao riješiti povećanjem stranice sa 8kB na veću veličinu.

Važan test bio je i mjerenje vremena koje je potrebno sustavu da unese jednu točku u bazu. S vremenom bi sustav pohranio veliku količinu točaka, ali prema rezultatima istraživanja vrijeme unosa ne utječe na performanse sustava do 400 milijuna točaka.

Literatura:

1. Paton, Norman, Williams, M.H., Dietrich, K., Liew, O., Dinn, A., Patrick, A. (2000) : „VESPA: A Benchmark for vector spatial databases“

2. Stonebraker, M., Frew, J., Gardels, K., Meredith, J. (1993) : „The Sequoia 2000 Benchmark“

3. Ray, S.,Simion, B,, Demke Brown, A. (2011) : „Jackpine: A Benchmark to Evaluate Spatial Database Performance“

4. Power, R. (2009) : „Testing geospatial database implementations for water data“

5. Suter, R. (2012) : „MongoDB - An introduction and performance analysis“

POPIS URL-ova:

URL 1: „How does a GPS tracking system work?“, http://www.eetimes.com/document.asp?doc_id=1278363*, 01.08.2014.*

URL 2: „Backitude“: http://www.appbrain.com/app/backitude-gps-location-tracker/gaugler.backitude*, 02.08.2014.*

URL 3: „Gis Objects“: http://postgis.net/docs/manual-2.1/using_postgis_dbmanagement.html#RefObject*, 02.07.2014.*

URL 4: „Using OpenGIS Standards“: http://postgis.net/docs/manual-2.1/using_postgis_dbmanagement.html#spatial_ref_sys*, 12.07.2014.*

URL 5: „Introduction to PostGIS: Section 17: Geography“, http://workshops.boundlessgeo.com/postgis-intro/geography.html *, 15.07.2014.*

URL 6: „Gegraphic Information Systems“, http://gis.stackexchange.com/questions/6681/what-are-the-pros-and-cons-of-postgis-geography-and-geometry-types *, 10.07.2014.*

URL 7: „PostGIS Geography Type“: http://postgis.net/docs/manual-2.1/using_postgis_dbmanagement.html#PostGIS_Geography*, 11.07.2014.*

URL 8: „Introduction to PostGIS: Section 14: Spatial indexing“, http://workshops.boundlessgeo.com/postgis-intro/indexing.html*, 01.08.2014.*

URL 9: „GiST: A Generalized Search Tree for Secondary Storage“, http://gist.cs.berkeley.edu/gist1.html*, 01.08.2014.*

URL 10: „Flask API“, http://flask.pocoo.org/docs/api/ *, 12.08.2014.*

URL 11: „Psycopg2“, https://github.com/psycopg/psycopg2*, 03.08.2014.*

URL 12: „jQuery“, http://jquery.com/ *, 01.07.2014.*

URL 13: „Leaflet“, http://leafletjs.com/index.html*, 02.08.2014.*

URL 14: „Mapbox“: https://www.mapbox.com/about/*, 01.08.2014.*

URL 15: „Flask Quickstart“, http://flask.pocoo.org/docs/quickstart/*, 20.07.2014.*

URL 16: “The geoJSON format specification“, http://geojson.org/geojson-spec.html*, 02.08.2014.*

URL 17: „JSONify“, https://github.com/kushalpandya/JSONify*, 17.07.2014.*

URL 18: „jQuery.getJSON()“, http://api.jquery.com/jquery.getjson/*, 03.08.2014.*

URL 19: „JavaScript Timing Events“, http://www.w3schools.com/js/js_timing.asp*, 02.08.2014.*

URL 20: „Planet.gpx“, http://wiki.openstreetmap.org/wiki/Planet.gpx*, 01.08.2014.*

URL 21: „Populating a database“:
http://www.postgresql.org/docs/9.1/static/populate.html *, 02.08.2014.*

URL 22: „Database Page Layout“: http://www.postgresql.org/docs/9.0/static/storage-page-layout.html , 06.08.2014.

URL 23: „TOAST“: http://www.postgresql.org/docs/8.0/static/storage-toast.html, 06.08.2014.

POPIS SLIKA

POPIS TABLICA

POPIS GRAFOVA

PRILOG

1. btwebgis.py python skripta

```
from flask import Flask, request, render_template, jsonify, make_response, json, g
import psycopg2
import os
from datetime import datetime
app = Flask(__name__, static_url_path='')
@app.before_request
def before_request():
  try:
    g.conn = psycopg2.connect("dbname=dipl_backtitude user=ivkolar")
    g.conn.set_isolation_level(0)
  except:
    print "Cannot connect to db"
@app.teardown_request
def teardown_request(exception):
  conn = getattr(g, 'conn', None)
  if conn is not None:
    conn.close()

@app.route('/', methods = ['POST'])
def add_point():
    lon=request.form['longitude']
    lat=request.form['latitude']
    print 'longitude: ' + request.form['longitude'] + ' latitude: ' + request.form['latitude']
    cur = g.conn.cursor()
    sql = '''
    INSERT INTO points (geom,the_time)
    VALUES (ST_GeomFromText(%s,-1),(%s))'''
    params = ["POINT(" + lat + " " + lon + ")", datetime.now()]
    cur.execute(sql, params)
    g.conn.commit()
    cur.close()
    return '',201

@app.route('/getpoint', methods = ['GET'])
def get_point():
        cursor = g.conn.cursor()
        sql = 'SELECT ST_AsGeoJSON(geom) from points ORDER BY id DESC LIMIT 1'
        cursor.execute(sql)
        result = cursor.fetchall()
        mypoint = str(result)[3:][:-4]
        mygeojson=jsonify({"type":"Feature","geometry":json.loads(mypoint)})
        return mygeojson

@app.route('/map', methods = ['GET'])
def return_map():
        return render_template('map.html')


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=15201, debug=True)
```

2. gpx2csv.py skripta za prijenos točaka iz gpx datoteka u csv datoteku

```
import os
import urllib2
import psycopg2
import fnmatch
import re
import time
import csv

startDir="E:\\Diplomski radno\\germany\\gpx-planet-2013-04-09\\trackable\\000\\051"
c = csv.writer(open("germany.csv", "wb"), delimiter=";")

def getAllGPXFiles(startDir):
    f = [os.path.join(dirpath, f)
        for dirpath, dirnames, files in os.walk(startDir)
        for f in fnmatch.filter(files, '*.gpx')]
    return f

listAllFiles = getAllGPXFiles(startDir)
listData =[]
current_track = None
current_file = None
seq = 1
i = 0

while i < len(listAllFiles):
   f = open(listAllFiles[i])
   current_file = listAllFiles[i][-13:-4]
   lines = f.readlines()
   for item in lines:
      if item.find("<name>") != -1:
         start = item.find(">")
         end = item.find("/")
         current_track = item[start+1:end-1]
         seq=0

      elif item.find("<trkpt") != -1:
         startLat = item.find("lat=")
         endLat = item.find(" lon")
         endLon = item.find(">")
         lat= item[startLat+5:endLat-1]
         lon= item[endLat+6:endLon-2]
         seq=seq+1

         c.writerow([lat, lon, seq, current_track, current_file])
   if(i % 10000):
      print str(seq) + " " + lat + " " + lon + " " + " " + current_track + " "+ " "+ current_file
   i=i+1
c.close()
```

3. map.html predložak za kartu

```
<!doctype html>
<html lang="en">
<head>
  <link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.css" />

  <style>
    #map {
                width: 100%;
                height: 100%;
                position: absolute;
      z-index: :0;
    }

  </style>
  <title>Backitude Tracker WebGIS</title>
  <script src="http://cdn.leafletjs.com/leaflet-0.7.2/leaflet.js"></script>
  <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>


</head>

<body>
  <div id="map"></div>
    <script type="text/javascript">

      var pos;
      var latitude;
      var longitude;
      var map = L.map('map').setView([44.70,16.45], 14);
      L.tileLayer('https://{s}.tiles.mapbox.com/v3/ivkolar.i52hdgf2/{z}/{x}/{y}.png',
{attribution: 'Diplomski rad', maxZoom: 18}).addTo(map);


      function update_position() {
      $.getJSON("http://geoserver.geof.unizg.hr:15201/getpoint", function(data) {
        var latitude = data["geometry"]["coordinates"][0];
        var longitude = data["geometry"]["coordinates"][1];
      map.setView(new L.LatLng(latitude,longitude));
        if (!pos) {
            pos = L.marker([latitude,longitude]).addTo(map);
        }
        pos.setLatLng([latitude,longitude]).update();

    });
}
      setInterval(update_position, 2000);
      update_position();

    </script>
</body>
</html>
```

ŽIVOTOPIS

EUROPEAN
CURRICULUM VITAE
FORMAT

OSOBNE OBAVIJESTI

Ime	**KOLAR Ivan**
Adresa	**Mala Erpenja 45, 49217 Krapinske Toplice, Hrvatska**
Telefon	**098 9437184**
Faks	**-**
E-pošta	**ivkolar@geof.hr**

Državljanstvo	Hrvatsko
Datum rođenja	10.08.1990.

RADNO ISKUSTVO

• Datum (od – do)	Srpanj 2013 -
• Naziv i sjedište tvrtke zaposlenja	Geo-bt d.o.o. , Lug Zabočki /E, Zabok
• Vrsta posla ili područje	Praktična geodezija
• Zanimanje i položaj koji obnaša	Studentski posao
• Osnovne aktivnosti i odgovornosti	Terenski rad i izrada elaborata

ŠKOLOVANJE I IZOBRAZBA

• Datum (od – do)	2012. – .
• Naziv i vrsta obrazovne ustanove	Geodetski fakultet Zagreb
• Osnovni predmet /zanimanje	Diplomski studij geodezije i geoinformatike

• Datum (od – do)	2009. – 2012.
• Naziv i vrsta obrazovne ustanove	Geodetski fakultet Zagreb
• Osnovni predmet /zanimanje	Preddiplomski studij geodezije i geoinformatike
• Naslov postignut obrazovanjem	Prvostupnik geodezije i geoinformatike

• Datum (od – do)	2005. – 2009.
• Naziv i vrsta obrazovne ustanove	Gimnazija A.G. Matoša Zabok
• Osnovni predmet /zanimanje	Prirodoslovno - matematički smjer

OSOBNE VJEŠTINE I SPOSOBNOSTI

MATERINSKI JEZIK	**HRVATSKI JEZIK**
DRUGI JEZICI	
	ENGLESKI JEZIK
• sposobnost čitanja	IZVRSNO
• sposobnost pisanja	DOBRO
• sposobnost usmenog izražavanja	IZVRSNO
SOCIJALNE VJEŠTINE I SPOSOBNOSTI	KOMUNIKATIVNOST
ORGANIZACIJSKE VJEŠTINE I SPOSOBNOSTI	ORGANIZIRANOST TIMSKI RAD
TEHNIČKE VJEŠTINE I SPOSOBNOSTI	QUANTUM GIS, GRASS GIS, RAD S BAZAMA PODATAKA (POSTGRESQL, POSTGIS), AUTOCAD MAP, ZWCAD, OCAD, GEOMEDIA, IDRISI KILIMANJARO, GOOGLE SKETCHUP, COREL DRAW, INKSCAPE, THERMACAM RESEARCHER, TNTMIPS, MS OFFICE, OPEN OFFICE.
VOZAČKA DOZVOLA	B kategorija