Prostorno-vremenske baze podataka

Domiter, Monika

Master's thesis / Diplomski rad

2023

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

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:717195

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Monika Domiter

PROSTORNO-VREMENSKE BAZE PODATAKA

Diplomski rad

Voditelj rada:
prof. dr. sc. Robert Manger

Zagreb, 2023

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. _____

Sadržaj

Uvod

U ovom diplomskom radu biti će opisane prostorno-vremenske baze podataka i primjene istih u različitim područjima. U posljednje vrijeme došlo je do iznimno velikih pomaka u tehnološkom napretku. Najveći napredak očituje se na područjima informacijskih i komunikacijskih tehnologija što je dovelo do generiranja golemih količina podataka. U mnogo slučajeva, tim podacima pridružene su prostorne i vremenske odrednice. Neki od primjera takvih podataka su kretanje taksi vozila, objave na društvenim mrežama, migracije životinjskih vrsta te moždane aktivnosti. Kako bi se pojednostavila obrada tih podataka postoje algoritmi koji su specijalizirani za rad s takvim podacima.

U prvom i drugom poglavlju rada opisat ćemo teorijske pojmove koji su nužni za rad s prostorno-vremenskim bazama podataka. Prvo ćemo se upoznati s prostornim i vremenskim bazama podataka te njihovim svojstvima, a zatim to znanje objediniti kroz termin prostorno-vremenskih baza podataka. Prostorno-vremenski podaci koriste se u mnogim znanostima pa ćemo opisati primjenu istih.

U trećem poglavlju predstavit ćemo FlexTrack sustav koji korisnicima omogućuje postavljanje upita nad trajektorijama korištenjem novog upitnog jezika. Predstavit ćemo i dva algoritma koji izvrednjavaju te upite.

U posljednjem četvrtom poglavlju, odnosno u praktičnom dijelu izradit ćemo prostorno-vremensku bazu podataka koja reprezentira kretanje taksi vozila te ćemo pomoću tih podataka kreirati kartu na kojoj je prikazana vožnja taksija. Prostorno-vremenska baza podataka implementirana je u MySQL Workbenchu [1], a programsko rješenje izvedeno je korištenjem programskog jezika Python[2]. Podaci koji se obrađuju i koriste u izradi ovog rada preuzeti su s Kaggle platforme [3].

[1] `https://www.mysql.com/products/workbench/`

[2] `https://www.python.org/`

[3] `https://www.kaggle.com/datasets/crailtap/taxi-trajectory`

Poglavlje 1

Osnove prostornih i vremenskih baza podataka

Kako bismo shvatili pojam prostorno-vremenskih baza podataka, potrebno je najprije definirati pojam baza podataka.

Baza podataka skup je međusobno povezanih podataka pohranjenih u vanjskoj memoriji računala. [3] Dakle, to je kolekcija podataka pohranjenih na računalu na sistematski način tako da računalni program može poslati upit bazi podataka na koji ona odgovara. One služe za bolju dostupnost i razvrstavanje podataka. Podaci su istovremeno dostupni raznim korisnicima i aplikacijskim programima.

Sustav za upravljanje bazom podataka - SUBP (Data Base Managament System - DBMS) je poslužitelj (server) baze podataka koji omogućava upravljanje podacima u bazi kao što je upisivanje, promjena, brisanje i čitanje podataka. On u ime klijenta obavlja sve operacije s podacima te brine za sigurnost podataka i automatizira administrativne poslove s bazom.

Kao što i sam naziv govori, prostorno-vremenske baze podataka možemo podijeliti na prostorne i vremenske baze podataka pa zbog toga u ovom poglavlju govorimo nešto više o njima.

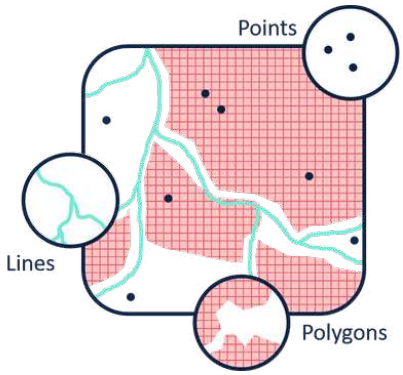
1.1 Prostorne baze podataka

Prostorne baze podataka su baze podataka koje pohranjuju prostorne objekte. Prostorni podaci su podaci koji sadrže geografske informacije. Te geografske informacije mogu definirati lokaciju, prostorni opseg, oblik i sl. Lokacija uglavnom sadrži geografske koordinate odnosno geografsku dužinu i širinu, a prostorni opseg predstavlja x i y koordinate kutova rastera u geografskom prostoru. U geografiji postoji princip koji se zove Toblerov prvi zakon geografije koji kaže da je ”Sve je povezano sa svime, ali bliske stvari su

bolje povezane od udaljenih stvari". Ovaj zakon temelj je koncepata prostorne ovisnosti i prostorne autokorelacije. Upravo iz tog zakona dolazi prva karakteristika prostornih podataka, a to je ovisnost. Druga bitna karakteristika prostornih podataka je heterogenost. Ona dolazi iz činjenice da svaka lokacija ili područje ima svoju jedinstvenu prirodu koja pridonosi heterogenosti u prostornim podacima. Uzimajući u obzir činjenicu da se neki geografski događaji dešavaju češće na jednom mjestu nego na drugom možemo reći da se prostorna heterogenost odnosi na neravnomjernu raspodjelu vrijednosti varijable u prostoru. Općenito se pripisuje krajoliku ili populaciji. Postoji puno tipova i načina podjele prostornih podataka. Dva važna tipa prostornih podataka su geometrijski i geografski podaci. Geometrijski tip podataka služi za prikazivanje informacija u dvodimenzionalnom prostoru gdje je položaj točke na plohi definiran koordinatama x i y. Google Maps je aplikacija koja koristi geometrijski tip podataka. Geografski tip podataka služi za spremanje geodetskih prostornih podataka kojima je vidljiva i zakrivljenost zemlje. Poznati primjer geografskih podataka je sustav globalnog pozicioniranja (GPS).

Prostorni objekt je digitalna reprezentacija objekata stvarnog svijeta s pridruženim atributima. Ti objekti ili imaju barem jedan prostorni atribut ili sadrže prostornu informaciju koja se odnosi na cijeli objekt. Prostorni atribut prostornog objekta je atribut koji sadrži informaciju o geografskoj poziciji objekta u stvarnom svijetu. On daje informacije obzirom na prostorne lokacije, npr. geografsku širinu, geografsku dužinu, nadmorsku visinu itd. Moramo imati na umu da vrijednost prostornog atributa uvijek mora biti u domeni prostornih podataka.

Kako bi stvarni svijet predstavili što jednostavnije koriste se sljedeći apstraktni tipovi prostornih podataka: točka, linija i regija/poligon.



Slika 1.1: Tri osnovna apstraktna tipa prostornih podataka

Točka predstavlja objekt za koji je relevantan samo njegov položaj u prostoru,ali ne i njegov opseg. Na primjer, u modelu koji opisuje veliko geografsko područje grad se može modelirati kao točka. Linija, koju u ovom kontekstu treba shvatiti kao krivulju u prostoru,

je osnovna apstrakcija za objekte za kretanje u prostoru, odnosno za veze u prostoru (ceste, rijeke, kablovi za struju, telefon,...). Regija je apstrakcija za nešto što ima opseg u 2D prostoru, npr. državu, jezero, pokrajinu, kontinent ili nacionalni park. Slika 1.1. prikazuje tri osnovna apstraktna tipa prostornih podataka. Za učinkovitije predstavljanje stvarnog svijeta moramo definirati i složenije prostorne podatke. Jedan od složenih prostornih podataka je 'pokretna točka' koja reprezentira trajektorije. O trajektorijama će biti viši rečeno u trećem poglavlju.

Podsjetimo se da Toblerov prvi zakon geografije govori da je sve povezano sa svime. Jack Dangermond jedna je od važnih osoba u razvoju i evoluciji geografskih informacijskih sustava (GIS). On govori da je GIS inteligentan živčani sustav za zemlju. Ove stvari nas potiču da definiramo prostorna vezu. Prostorna veza je ona koja se odnosi na dva prostorna objekta. Objasnimo to sljedećim jednostavnim primjerom. *Lošinj se nalazi južno od Cresa* definira prostornu vezu *južno od* između dva prostorna objekta, *Lošinja* i *Cresa*. Prostorne veze možemo podijeliti na tri tipa: []

- *Topološke veze* - Topološke veze povezuju dva prostorna objekta na temelju postavljenog odnosa. Mogu se podijeliti na tri osnovna tipa: povezanost, susjedstvo i ograđenost. Povezanost opisuje kako su povezane linije. Susjedstvo opisuje jesu li dva područja jedno kraj drugog, i ograđenost opisuje jesu li dva područja ugniježđena odnosno nalazi li se jedno u drugome. *Jednako*, *disjunktno*, *siječe*, *sadrži* i *susjedan* neki su od primjera.

- *Veze smjera i orijentacije* - Lokacija nam govori gdje se objekt nalazi u odnosu na unaprijed određen referentni sustav, tj. globalno ishodište i skup osi koordinatnog sustava. Orijentacija objekta je jednako važna. Smjer može relativan ili apsolutan. Smjerovi poput lijevo/desno, gore/dolje i naprijed/natrag su relativni, dok su smjerovi sjever/jug i istok/zapad primjeri apsolutnog smjera.

- *Veze udaljenosti* - Ove veze između prostornih objekata definirane su koristeći udaljenost između prostornih objekata. Udaljenosti mogu biti subjektivne kao npr. blizu, daleko itd ili objektivne kao 20 km itd. Ove mjere definiraju oblik i površinu, odnosno veličinu prostornih objekata. Zahvaljući tome dva prostorna objekta možemo usporediti pomoću relacija *veći/manji*, *duži/kraći* itd.

1.2 Vremenske baze podataka

Baze podataka uglavnom spremaju informacije o trenutnom stanju, a ne o prošlom stanju. Na primjer u bazi zaposlenika neke kompanije ako se plaća ili adresa određene osobe mijenja, baza podataka se ažurira i stari podatak se više ne pamti, on više ne postoji u bazi

podataka. Međutim za mnoge aplikacije važno je imati prošle vrijednosti i vrijeme u kojem su podaci ažurirani. Odnosno, potrebno je znanje o evoluciji tih podataka. Upravo su tu vremenske baze podataka od iznimne koristi. Također, većina organizacija upravlja podacima koji se mijenjaju tijekom vremena. Od vas se može tražiti da pokažete kako su vaši podaci izgledali na određeni datum i sl. To nećete moći učiniti bez vremenske baze podataka. Dakle, vremenske baze podataka su baze podataka koje imaju ugrađene vremenske aspekte radi omogućivanja rada s takvim podacima putem prilagođenog i proširenog jezika. Vremenski podaci su podaci koji u sebe ugrađuju vremensku informaciju, a vremenski objekt je instanca koja ima barem jedan vremenski atribut. Vremenski atribut je atribut koji sadrži vremensku vrijednost.

Postoje različit aspekti modeliranja vremena u podacima. Sada ih navodimo i objašnjavamo.

- *Vremenski tipovi podataka* - Postoje dva apstraktna vremenska tipa podataka - točka i interval. Točku koristimo kada se neki događaj dogodio u točno jednom trenutku i kada to vrijeme događaja želimo spremiti. Neki primjeru su: ulazak u trgovinu u 14:05, utakmica počinje u 18 i slično. U 14:0 i u 18 odnose se na točnu točku u vremenu u kojoj se događaj dogodio. Interval koristimo kada događaj traje kroz neki vremenski period. Kada koristimo interval možemo spremiti početno i završno vrijeme događaja. *Ana se vozila autocestom od 18 do 20:25*, *Ispit je trajao od 10 do 12*, *Jučer su završili radovi na cesti* neki su od primjera intervalnog tipa podatka. Od 10 do 12, od 18 do 20:25 i jučer odnose se na interval u kojem se događaj dogodio.

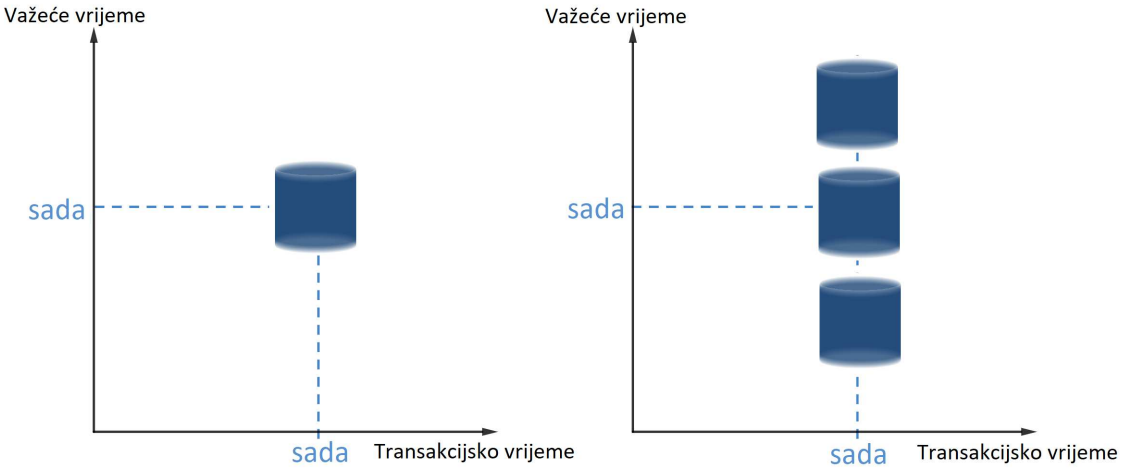
- *Vremenski poredak* - Važna karakteristika vremena je da je ono linearno i prirodno uređeno. Kada je potrebno koristi se i razgranati odnosno nelinearni model vremena. Za razliku od linearnog modela koji koristi jednu vremensku crtu te daje potpuni poredak među događajima, nelinearni model pruža djelomični poredak među događajima.

- *Redoslijed događaja/Transakcije* - Uključivanje vremena u baze podataka od velike je koristi. Ono uvodi prirodni poredak ili strukturira događaje u bazi podataka. Kako bi se podržao vremenski poredak među događajima podaci mogu imati vremenski žig (apsolutno vrijeme) ili neka vrsta relativnog redoslijeda (relativno vrijeme) može biti prožeta među događajima. *16/09/2022T10:51:23* primjer je vremenske oznake dok su *prije, nakon, simultano* primjeri koji daju relativni poredak među događajima.

- *Vremenska prošlost* - Podacima se vremenska oznaka može pridružiti na razini atributa, razini retka ili na razini relacije. Vremensko označavanje može se izvršiti pomoću važećeg vremena ili transakcijskog vremena. Važeće vrijeme (engl. valid time) je vrijeme u stvarnom svijetu kada se neki događaj dogodio ili kada je neka činjenica važeća dok je transakcijsko vrijeme (engl. transaction time) vrijeme kada

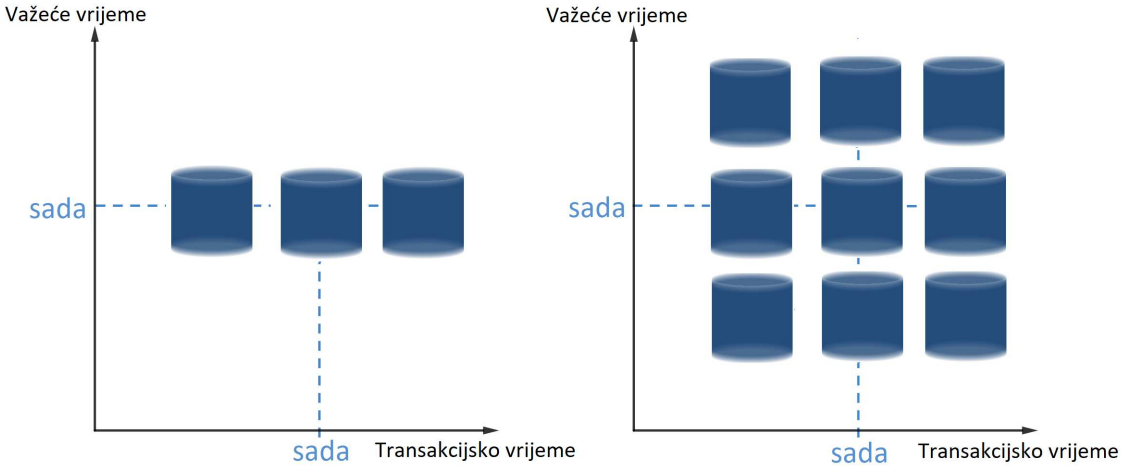
je događaj zabilježen u bazi podataka ili vremenski interval tijekom kojeg se baza podataka nalazi u određenom stanju. Može se koristiti i bitemporalno vrijeme. To je vrijeme kojeg čini kombinacija važećeg i transakcijskog vremena. Važeće vrijeme može biti tipa točke ili intervala, dok je transakcijsko vrijeme uvijek u oblike točke.

- Ovisno o tome podupire li baza podataka važeće vrijeme ili transakcijsko ili oba, možemo ih podijeliti u četiri kategorije.
  - *Trenutačne baze podataka (engl. snapshot database)* - One čuvaju najnoviju verziju podataka. Tijekom promjene stanja baze, staro stanje baze se zaboravlja te se pamti zadnje uneseno stanje. Dakle, one opisuju jedan trenutak u stvarnom svijetu, najčešće sadašnjost.
  - *Povijesne baze podataka* - Ove baze pohranjuju povijest podataka o stvarnom svijetu. Kako podržavaju samo važeće vrijeme, one pohranjuju stanje podataka duž osi važećeg vremena. Tijekom promjene stanja baze, stara vrijednost se i dalje briše.



Slika 1.2: Spremanje podataka u snapshot (lijevo) i povijesnoj bazi podataka (desno)

  - *Rollback baze podataka* - Ove baze podataka pohranjuju izmjene u bazi podataka. Kako podržavaju samo koncept transakcijskog vremena, pohranjuju stanje baze podataka duž osi transakcijskog vremena. U ove baze podaci se samo dodaju, tj. niti jedan podatak se fizički ne briše.
  - *Bitemporalne baze podataka* - Bitemporalne baze su zapravo kombinacija povijesnih i rollback baza podataka. One podržavaju i važeće i transakcijsko vrijeme pa pohranjuju stanje baze podataka uz obje vremenske osi.



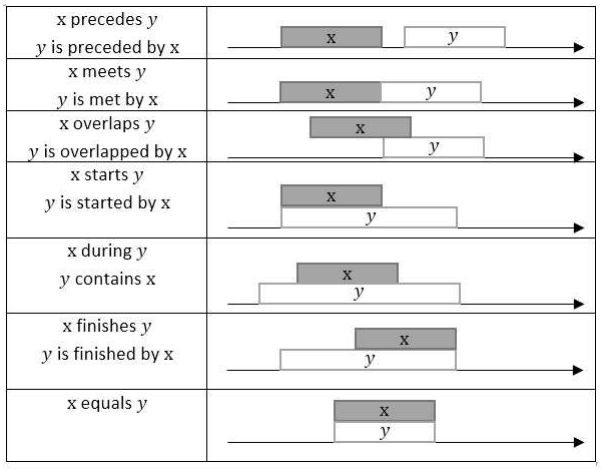
Slika 1.3: Spremanje podataka u rollback (lijevo) i bitemporalnoj bazi podataka (desno)

- *Vremenska domena ili struktura* - Vrijeme možemo modelirati ovisno o tome je li temeljni proces diskretni ili neprekidni fenomen. Temeljni proces određuje je li vremenska domena diskretna ili neprekidna. Diskretni fenomen predstavlja događaje koji se događaju u stvarnom svijetu u diskretnom vremenu. Podaci se stoga prikupljaju kako i kada se događaji dogode. Diskretni modeli izomorfni su prirodnim ili cijelim brojevima gdje svaki prirodan ili cijeli broj odgovara osnovnoj mjernoj jedinici vremena - *chrononu*. Chronone možemo grupirati i u veće jedinice vremena, npr. sati, dani, mjeseci... Kada je temeljni proces neprekidni fenomenom tada prikupljene informacije o objektima koji se neprekidno mijenjaju dovode do velike vremenske gustoće. Mjerenje vremenskih uvjeta kao što su temperatura i padalina je neprekidno. Kontinuirani modeli izomorfni su realnim brojevima gdje svaki realan broj odgovara jednom vremenskom trenutku.

- *Vremenska skala ili Zrnatost* - Vremenska zrnatost definira periodičnost prikupljanja podataka. Na primjer *Baza podataka za temperaturu grada Zagreba ažurira se svakih 30 minuta* definira vremensku zrnatost od 30 minuta.

- *Vremenski odnosi* - Vremenski odnosi definiraju vezu između dva vremenska entiteta. Allenova intervalna algebra [1] definirana je skupom od 13 vremenskih predikata koji predstavljaju mogući odnos između dva vremenska intervala. Oni su ekvivalentni topološkim odnosima između prostornih objekata. Obzirom na dva vremenska intervala, x i y, Allenova intervalna algebra prikazana je na slici 1.4. Neki od vremenski predikata su *prethodi*, *preklapa* i *tijekom*. Druga vrsta vremenskih odnosa

definirana je vremenskim rasponom. Taj raspon ekvivalentan je metričkim odnosima između prostornih objekata.



Slika 1.4: 13 mogućih odnosa prema Allenu

- *Vremenska reprezentacija* - Od davnina postoje različiti prikazi vremena i kao takvi mogu se koristiti u bazama podataka. Primjer uključuje Greenwich Mean Time (GMT). Također prikazi i mjerenja vremena ovise o subjektivnim konstruktima koji variraju prema promjena u našim vjerovanjima, društvenim potrebama i tehničkom napretku. Neki od primjera su akademski kalendar, sjetveni kalendar i financijski kalendar.

Poglavlje 2

Prostorno-vremenske baze podataka

Nakon što smo se upoznali s prostornim i vremenskim bazama podataka možemo to znanje objediniti kroz termin prostorno-vremenskih baza podataka. Dakle, prostorno-vremenske baze podataka su baze koje pohranjuju podatke s prostornim i vremenskim atributima.

2.1 Opis i primjena

Prostor i vrijeme sveprisutni su aspekti promatranja u mnogim područjima. Neka od tih područja uključuju društvene mreže, epidemiologiju, promet, neuroznanost i klimatologiju. Upravo se zbog toga u današnje vrijeme prostorno-vremenskim podacima pridodaje sve veća pažnja, oni se sve više prikupljaju, obrađuju i proučavaju. Za svako područje ukratko opisujemo izvore prostorno-vremenskih podataka i motivaciju za analizu prostorno-vremenskih podataka.

- *Društvene mreže* - Korisnici društvenih mreža kao što su Facebook, Twitter i Instagram svakodnevno objavljuju svoja iskustva i doživljaje na određenom mjestu i u određeno vrijeme. Koristeći ove podatke može se proučavati kolektivno korisničko iskustvo na određenom mjestu za određeno vremensko razdoblje. Osim toga, korištenjem ovih podataka odnosno na temelju objava korisnika, može se pratiti širenje bolesti kao što je gripa, proučavati širenje politički pokreta i automatski se mogu otkriti događaji kao što su potresi, požari i sl.

- *Epidemiologija* - Elektronički zdravstveni karton (EZK) sadrži skup medicinskih podataka o pacijentu nužnih za kvalitetno provođenje zdravstvene njege. Takvi podaci koji se pohranjuju po bolnicama pružaju demografske podatke o pacijentima te dijagnoze postavljene na pacijentima u različitim vremenskim trenutcima. Ovi podaci se proučavaju kako bi se otkrili prostorno-vremenski obrasci za različite bolesti te kako bi se otkrili načini širenja epidemije.

- *Promet* - Početci taksi prijevoza kreću još u 17. stoljeću, međutim dolaskom tehnologije taksi tvrtke nude mobilne aplikacije za pozivanje taksija. Većina taksija danas je opremljena GPS uređajima koji su važan senzorski uređaj za proučavanje kretanja ljudi i aktivnosti. Podaci skupljeni GPS-om sadrže informacije o svakom putovanju korisnika taksi vozilom. Te informacije sadrže vrijeme i lokaciju ukrcaja i iskrcaja te lokaciju za svaku sekundu tijekom vožnje. Ovi podaci mogu nam pomoći razumjeti kako se građani kreću prostorno kroz vrijeme te kakav je utjecaj prometa i vremenskih neprilika. Također ti se podaci mogu koristiti za istraživanje dinamike prometa na temelju zajedničkih obrazaca kretanja taksija.

- *Neuroznanost* - Neuroznanost proučava moždane procese koji su temelj ljudskog ponašanja. Neuralna aktivnost snima se korištenjem različitih tehnologija kao što su funkcionalna magnetska rezonanca (fMRI), elektroencefalografom (EEG) i magnetoencefalografija (MEG). Prostorna i vremenska rezolucija neuralne aktivnosti razlikuje se za svaku od tehnologija. Neuralna aktivnost mjeri se na milijunima lokacija koristeći fMRI, a koristeći EEG mjeri se samo na desetak lokacija. Također, fMRI mjeri aktivnost svake dvije sekunde, a vremenska rezolucija EEG podataka je uglavnom 1 milisekunda. Ovi podaci se proučavaju kako bi se mogli odrediti poremećaji u odnosu na normalne funkcije. Otkrivanje poremećaja može se iskoristiti za unapređenje dijagnostičkih postupaka i razvoj terapijskih tehnika i vještina.

- *Klimatologija* - Klimatologija je znanost o klimi. Ova znanost prikuplja i obrađuje podatke koji se odnose na povijesne i trenutne atmosferske uvjete kao što je temperatura zraka, gustoća, tlak zraka, vlaga i sl. Ovi podaci se proučavaju kako bi unaprijedili naše razumijevanje prirodnih pojava te kako bi se bolje pripremili za nepovoljne uvjete pravovremenim informiranjem o mjerama zaštite.

Iako smo naveli samo dio područja u kojima se koriste prostorno-vremenski podaci te njihovu primjenu, treba imati na umu da postoje još mnoga. Ovi podaci koriste se u biologiji kako bi se proučavalo kretanje životinja te preseljenje vrsta i njihovo izumiranje, u šumarstvu kako bi se planirala sadnja i sječa drveća, u geofizici za predviđanje vulkanskih aktivnosti, u ekologiji za praćenje onečišćenja. Agencija Europske unije za suradnju u provođenju zakona, koja je poznata pod imenom Europol je agencija za borbu protiv kriminala i provođenje zakona Europske unije. Jedan od njenih zadataka je pohranjivanje informacija o prijavljenim zločinima. Ti podaci sadrže mjesto i vrijeme zločina te vrstu zločina, odnosno radi li se o napadu, provali, krađi, vandalizmu itd. Koristeći dobivene podatke mogu se provesti studije o utjecaju zakona na stopu smanjenja kriminala.

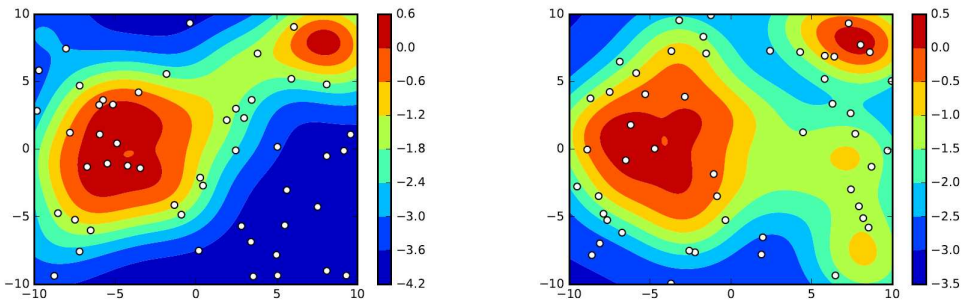
Postoji niz različitih tipova prostorno-vremenskih podataka koji se mogu pojaviti u različitim područjima. Tipovi prostorno-vremenskih podataka razlikuju se u načinu na koji se prostor i vrijeme koriste u prikupljanju i reprezentaciji podataka. Postoje četiri

uobičajena tipa prostorno vremenskih podataka: podaci o događajima, trajektorije, podaci o referentnim točkama i rasterski podaci. U nastavku opisujemo svaki tip prostorno-vremenskih podataka.

- *Podaci o događajima* - Prostorno-vremenski događaj može se reprezentirati koristeći lokaciju i vrijeme kada se događaj dogodio. Na primjer, prometna nesreća može biti opisana lokacijom nesreće i vremenskim trenutkom kada se nesreća dogodila. Svaki događaj osim lokacije i vremenske informacije može imati atribute koji nisu prostorno-vremenske prirode. Ti atributi daju dodatne informacije o događaju. Možemo zaključiti da se prostorno-vremenski događaji koriste i imaju primjenu u mnogim područjima. U epidemiologiji se širenje bolesti može prikazati pomoću mjesta i vremena gdje se pacijent zarazio, u kriminalistici se zločin također može okarakterizirati lokacijom zločina zajedno s vremenom kada se on dogodio.

- *Trajektorije* - Trajektorije sadrže položaje odnosno lokacije objekta u pokretu u određenim vremenskim trenucima. Primjena trajektorija je danas sve veća. Ljudi svakodnevno bilježe svoja kretanja u obliku trajektorija koristeći mobilne uređaje. Sve je više vozila kao što su taksiji, autobusi i zrakoplovi opremljeno GPS uređajima. Podaci u obliku trajektorija najčešće se prikupljaju GPS senzorima koji omogućuju prijenos podataka o lokaciju tijekom vremena. Trajektorije imaju najveću primjenu u prometu, ali koriste se i u ekologiji, urbano planiranju i poslovanju.

- *Podaci o referentnim točkama* - Podaci o referentnim točkama sastoje se od mjerenja neprekidnog prostorno-vremenskog polja kao što je temperatura, vegetacija ili stanovništvo nad skupom pokretnih referentnih točaka u prostoru i vremenu.

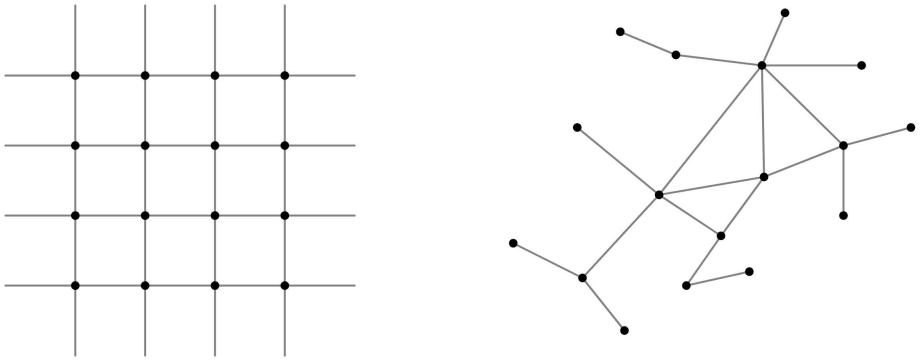


Slika 2.1: Referentne točke (bijeli kružići) u dva različita vremenska trenutka

Na primjer, meteorološke varijable kao što su atmosferski tlaka, temperatura, vlažnost te brzina i smjer vjetra mjere se korištenjem meteoroloških balona. To su meteorološki instrumenti koji lebde u svemiru te kontinuirano prate i bilježe vremensko stanje na točkama u prostoru i vremenu. Slika 2.1 prikazuje referentne točke u

dva različita vremenska trenutka. Trake s bojama prikazuju distribuciju prostorno-vremenskog polja u dva vremenska trenutka. Sljedeći primjer su senzori na plutačama, odnosno meteorološko-oceanografskim plutačama koji neprekidno mjere oceanografske parametre kao što su temperatura, slanost, razina kisika i pH vrijednost na lokacijama koje se mijenjaju tijekom vremena. U oba primjera konačan uzorak prostorno-vremenskih referentnih točaka koristi se kako bi se predstavilo ponašanje neprekidnog prostorno-vremenskog polja. Mjerenja s balona i plutača omogućit će bolje predviđanje vremenskih promjena te bolje poznavanje ekosustava.

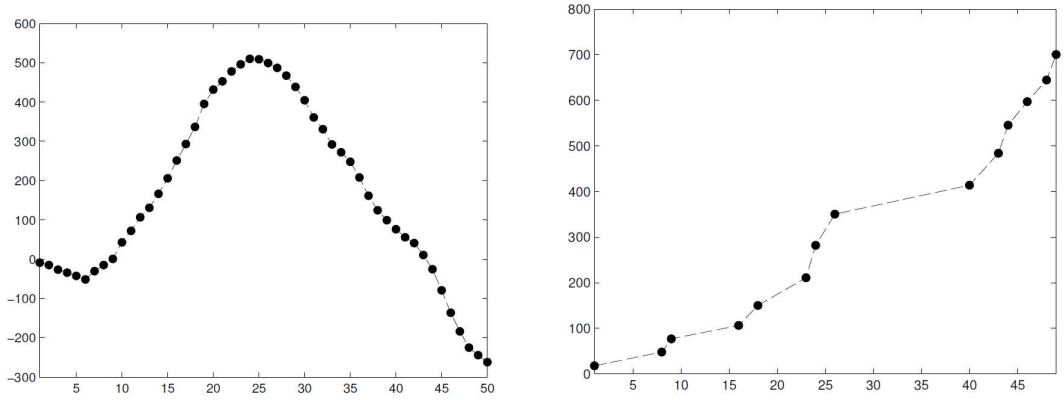
- *Rasterski podaci* - U rasterskim se podacima mjerenja diskretnog ili neprekidnog prostorno-vremenskog polja bilježe na fiksnim lokacijama u prostoru i u fiksnim vremenskim trenucima. Vidimo da je ovo suprotnosti u odnosu na podatke o referentnim točkama gdje referentne točke mogu mijenjati svoju lokaciju kroz vrijeme te prikupljati podatke u različitim vremenskim trenucima. Formalno, neka je $\mathcal{S} = \{s_1, s_2, \ldots, s_m\}$ skup fiksnih lokacija koje mogu biti pravilno ili nepravilno raspoređene u prostoru (slika 2.2). Za svaku lokaciju bilježimo opažanja na fiksnom skupu vremenskih trenutaka $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ koji opet mogu biti pravilno raspoređeni s jednakim vremenskim odgodama ili nepravilno (slika 2.3).



Slika 2.2: Skup lokacija u rasteru pravilno (lijevo) i nepravilno (desno) raspoređenih u prostoru

Raster je Kartezijev produkt $\mathcal{S} \times \mathcal{T}$ koji rezultira potpunom prostorno-vremenskom mrežom, gdje svaki vrh na mreži ima različite vrijednosti mjerenja. Neki primjeri rasterskih podataka su fMRI video sekvence moždane aktivnosti i slike Zemljine površine prikupljene satelitima. Važno je napomenuti da dok neki primjeri prostorno-vremenskih raster podataka bilježe mjerenja u točkastim vrhovima (npr. mjerenja prikupljena mrežom senzora), drugi rade grupna mjerenja nad područjima u svakoj

ćeliji. Različite se demografske informacije prikupljaju grupno prema jedinicama lokalne ili područne (regionalne) samouprave kao što su gradovi, općine, županije itd. Još jedno važno svojstvo rasterskih podataka je rezolucija mreže (u prostoru i vremenu), koja se koristi za prikupljanje podataka. Svako područje znanosti koje koristi rasterski tip podataka koristi se različitim rezolucijama prostora i vremena. Na primjer fMRI tehnologija mjeri moždane aktivnosti na svakoj 1 mm $\times$ 1 mm $\times$ 1 mm lokaciji, dok EEG tehnologija mjeri aktivnost na odabranih desetak lokacija.



Slika 2.3: Skup lokacija u rasteru pravilno (lijevo) i nepravilno (desno) raspoređenih u vremenu gdje je na x-osi zabilježeno proteklo vrijeme, a na y-osi vrijednost mjerenja

2.2 Rudarenje u prostorno-vremenskim bazama podataka

Rudarenje podataka (eng. *data mininig*) proces je traženja novih, razumljivih, smislenih i potencijalno korisnih pravilnosti među velikim količinama podataka. Rudarenje podataka uglavnom se provodi na velikim količinama podataka iz baza podataka s ciljem otkrivanja korisnog znanja ili informacija. Rudarenjem se otkrivaju odnosi i logičnost te općenito bilo kakve strukture među podacima. Metoda se naziva rudarenje podataka zbog toga što se u velikim količinama podataka traže informacije koje ”vrijede zlata”.

Rudarenje prostorno-vremenskim podacima netrivijalan je postupak kojim pronalazimo nove i korisne informacije iz prostorno-vremenskih baza podataka. Zapravo bi mogli reći da je rudarenje prostorno-vremenskim podacima usredotočeno na traženje prostorno-vremenskih veza, tj. odnosa iz prostorno-vremenskih baza podataka. Postoji više razloga

zašto većina klasičnih algoritama za rudarenje podataka daje loše rezultate kada se primjene na prostorno-vremenske baze podataka. Navodimo ih u nastavku teksta.

- Jedna od glavnih pretpostavki koja se koristi u klasičnoj analizi je da su podaci međusobno neovisni i disjunktni. Međutim, kada analiziramo prostorno-vremenske podatke pretpostavka da su ti podaci nezavisni je netočna. Prostorno-vremenski podaci međusobno su povezani. Na primjer, ljudi sa sličnim interesima, zanimanjima i istim podrijetlom imaju tendeciju grupiranja zajedno u istim susjedstvima. Autokorelacija prostorno-vremenskih podataka rezultira koherentnošću prostornih promatranja i glatkoćom vremenskih promatranja.

- Sljedeća pretpostavka koja se koristi u klasičnim algoritmima za rudarenje podataka jest homogenost ili stacionarnost instanci. To implicira da svaka instanca pripada istoj populaciji te je stoga i identično distribuirana. Međutim, prostorno-vremenski podaci pokazuju heterogenost i u prostoru i vremenu. Poznato je da različite prostorne regije mozga obavljaju različite funkcije, pa stoga rezultiraju različitim fiziološkim odgovorima na podražaj.

- Prostorne-vremenske baze podataka često rezultat neprekidnog prirodnog procesa, dok su tradicionalne baze podataka diskretne.

- U prostorno-vremenskim bazama podataka obrasci su često lokalni, dok se u klasičnim algoritmima rudarenja podataka često fokusira na globalne obrasce.

- Klasični algoritmi rudarenja podataka ne podnose dobro rudarenje na većoj granularnosti prostora i vremena.

- Klasični algoritmi ne uključuju prostornu i vremensku semantiku kao što je topologija, udaljenost, prostorno-vremenska blizina itd.

- Svi klasični algoritmi za rudarenje podacima bazirani su na analizi potrošačke košare, odnosno analizi proizvoda koje kupac zajedno kupuje prilikom jedne kupovine. Međutim, proces rudarenja u prostorno-vremenskim bazama podataka uvelike se razlikuje od analize potrošačke košarice te je kompleksniji i zamršeniji.

Vidjeli smo da postoji mnogo razloga zašto korištenje klasičnih algoritama nije adekvatno te kako bi analizirali prostorno-vremenske podatke potreba za prilagođenim alatima za rudarenje postaje veliki izazov. Kako bi se efikasno rudarilo prostorno-vremenskim bazama podataka potrebne su mnoge preinake postojećih tehnika i alata. Potrebno je razviti različite strukture podataka za predstavljanje različitih tipova prostorno-vremenskih podataka. Kako su prostorno-vremenske baze podataka astronomske veličine potrebno je dizajnirati učinkovite tehnike indeksiranja i pretraživanja podataka. Također potrebno

je razviti učinkovit upitni jezik s bogatom semantikom koji podržava postavljanje upita na prostorno-vremenskoj bazi podataka. Jedan od tih jezika je upitni jezik fleksibilnih obrazaca (eng. *Flexible Pattern Query Language*) koji je detaljno predstavljen u idućem poglavlju.

Poglavlje 3

Trajektorije i FlexTrack sustav

3.1 Trajektorije

Razvoj informacijskih i komunikacijskih tehnologija (ICT) generirao je goleme količine podataka, posebno podatke u obliku trajektorija. Ti podaci sadrže specifične geografske lokacije i odgovarajuće vremenske oznake. Oni su snimljeni raznim uređajima koji koriste ugrađene senzore, audiofrekvencijsku identifikaciju (RFID), sustave automatizirane naplate karate, Globalni sustav za pozicioniranje (GPS), Globalni sustav za mobilne komunikacije (GSM) i ostalo. Stoga, trajektorije označavaju putanje objekata koji se kreću u prostoru kroz vrijeme. Neki primjeri trajektorija uključuju rutu kojom se taxi vozi od mjesta ukrcaja do mjesta iskrcaja ili obrasce migracije životinja koje se kreću radi boljeg pristupa hrani ili skloništu. Podaci u obliku trajektorija odavno su važno sredstvo proučavanja ljudskog ponašanja i rješavanja prometnih problema. Također, trajektorije su pokazale značajnu akademsku i praktičnu vrijednost. Istraživane su i analizirane kako bi se razvila rješenja za razna istraživačka pitanja kao što su prijevoz, urbano planiranje, otkrivanje abnormalnosti i prekršaja.

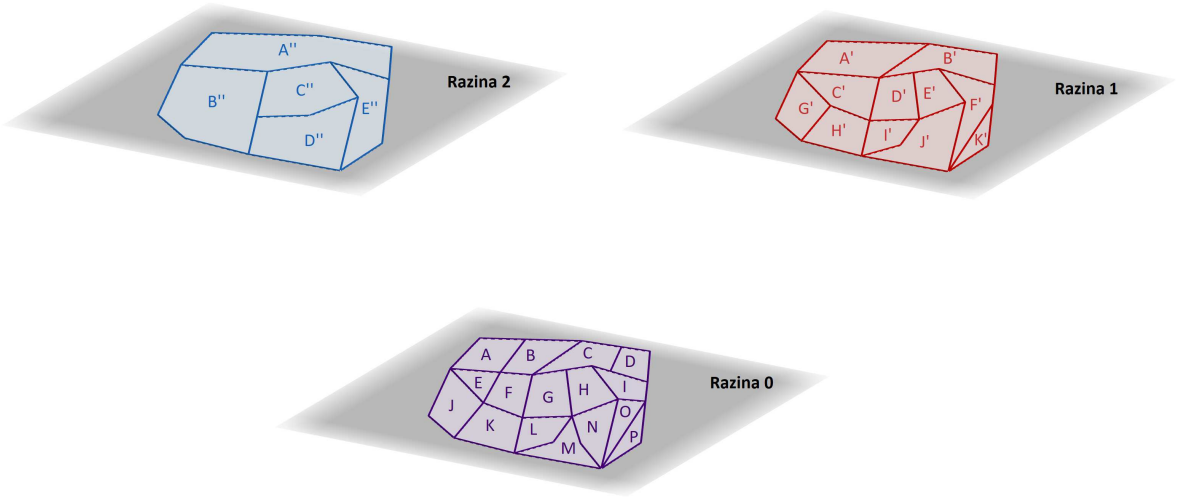
Dakle, velika upotreba i široka dostupnost mobilnih i lokacijskih uređaja omogućila je različitim aplikacijama spremanje podataka u obliku trajektorija. U tim aplikacijama svaka trajektorija ima jedinstveni identifikator i sastoji se od podataka o lokaciji za specifični objekt u pokretu tijekom uređenog niza vremenskih trenutaka. Obzirom da ove aplikacije generiraju veliku količinu podataka, potrebne su nove učinkovitije i djelotvornije tehnike za evaluaciju upita preko trajektorija. Postoje mnogi radovi i rješenja za vremenske nizove ili za podatke tokova događaja, ali oni nisu eksplicitno dizajnirani za rad s trajektorijama. Isto tako postoje radovi koji predstavljaju upitni jezik za rad s trajektorijama. Međutim, u tim upitima moguće je koristiti samo fiksirana područja. Zbog svih tih nedostataka, predstavlja se FlexTrack sustav koji pruža općenitiji i jak upitni okvir. FlexTrack sustav predstavili su M. R. Vieira, P. Bakalov i V. J. Tsotras 2010. godine. [5]

3.2 FlexTrack

Rudarenje prostorno-vremenskim bazama podataka zahtjeva mnoge promjene klasičnih tehnika i alata za rudarenje podacima. Jedna od bitnih promjena je uvođenje novog upitnog jezika koji omogućuje postavljanje upita na prostorno-vremenskoj bazi podataka. FlexTrack je sustav koji korisnicima omogućuje postavljanje upita na vrlo intuitivan način. On definira upitne obrasce (eng. *pattern queries*) kao regularne izraze nad konačnom prostornom abecedom. Upiti mogu sadržavati i fiksna i varijabilna područja te strukture regularnih izraza kao što su negacija, ponavljanje, opcionalne strukture i slično.

The Flexible Pattern Query Language

Trajektorija T_{id} objekta u pokretu O_{id} je uređen niz od w parova $(l_1, t_1), \ldots, (l_w, t_w)$ gdje je $l_i \in \mathbb{R}^d$ lokacija objekta zabilježena u vremenskoj oznaci t_i ($t_{i-1} < t_i$), $0 < i \leq w$. Takvi podaci prikupljaju se iz različitih aplikacija (npr. GPS, mobitel) i spremaju u bazu podataka. Praćeni objekti najčešće šalju svoju lokaciju uređaju za prikupljanje podataka pomoću paketa podataka koji sadrži njihov identifikator T_{id}, trenutnu lokaciju l_i i vremensku oznaku t_i. Svaka od aplikacija ima različitu primjenu i koristi se u različiti svrhe. Zbog toga objekti mogu javljati svoju lokaciju neprekidno ili samo u onim vremenskim trenucima kada se njihova lokacija promijeni. U FlexTrack sustavu prostorna je domena hijerarhijski podijeljena na razine. Na svakoj razini l prostorna je domena podijeljena fiksnim skupom Σ_l nepreklapajućih područja kao što se može vidjeti na slici 3.1. Važno je napomenuti da je svako područje na razini l formirano kao unija područja na prethodnoj razini, $l - 1$.



Slika 3.1: Skup područja definiran pomoću hijerarhije od 3 razine

Također, svojstvo nepreklapajućih područja odnosi se na područja na istoj razini, dok se područja s različitih razina mogu preklapati. Na primjer, područja A", B" i C" na razini 2 se ne preklapaju, dok se područje A" s razine 2 i B' s razine 1 preklapaju. Slika 3.1 ilustrira skup područja definiranih pomoću hijerarhije od 3 razine. Ovdje korisnik ima mogućnost definirati upite s finijom granularnošću abecede za dijelove od većeg interesa i većom granularnošću na ostalim dijelovima. Područja na koja dijelimo prostornu domenu odgovaraju područjima interesima. To mogu biti zračne luke, autobusni kolodvori, parkovi, trgovački centri, školski okruzi itd. Ta područja interesa tvore abecedu $\Sigma = \bigcup_l \Sigma_l = A, B, C, ...$ koju koristimo u našem upitnom jeziku.

Opći oblik upita izražen pomoću obrasca $Q = (\mathcal{S}\,[\bigcup \mathcal{D}])$ kombinacija je sekvencijalnog obrasca $\mathcal{S}$ i eventualno skupa ograničenja $\mathcal{D}$. Trajektorija se podudara s upitnim obrascem Q ako zadovoljava i $\mathcal{S}$ i $\mathcal{D}$. Komponenta $\mathcal{S}$ u obrascu odgovara nizu prostorno-vremenskih predikata $\mathcal{P}$ koji su specificirani pomoću područja iz Σ, a komponenta $\mathcal{D}$ predstavlja skup funkcija udaljenosti (npr. *Nearest Neighbour - NN* i njihove varijacije) i ograničenja (npr. @x!=@y, @z=A,B,C) koja mogu sadržavati područja definirana u $\mathcal{S}$. Obrazac $\mathcal{S}$ izražen je kao putanja (eng. *path expression*) koja se sastoji od proizvoljnog broja prostorno vremenskih predikata $\mathcal{P}$:

$$\mathcal{S} \rightarrow \mathcal{S}.\mathcal{S} \mid \mathcal{P} \mid !\mathcal{P} \mid \mathcal{P}^{\#} \mid ?^{+} \mid ?^{*} \qquad (\triangle)$$

"!" definira operator negacije, "#" izborni modifikator, "+" jedno ili više ponavljanja modifikatora, "*" nula ili više ponavljanja modifikatora i "?" zamjenski znak. U obrascu $\mathcal{S}$, zamjenski se znak "?" koristi za određivanje nebitnih dijelova u životnom vijeku trajektorije i može biti u dva oblika:

- "?+" : jedno ili više pojavljivanja bilo kojeg prostorno-vremenskog predikata (npr. $\mathcal{P}_{i-1}.?^{+}.\mathcal{P}_i$ znači da je predikat $\mathcal{P}_i$ zadovoljen nakon predikata $\mathcal{P}_{i-1}$ s jednim ili više posjećenih područja između njih), ili;
- "?*" : nijedno ili više pojavljivanja bilo kojeg prostorno-vremenskog predikata (npr. $\mathcal{P}_{i-1}.?^{*}.\mathcal{P}_i$ znači da predikat $\mathcal{P}_i$ može biti zadovoljen bilo kad nakon predikata $\mathcal{P}_{i-1}$).

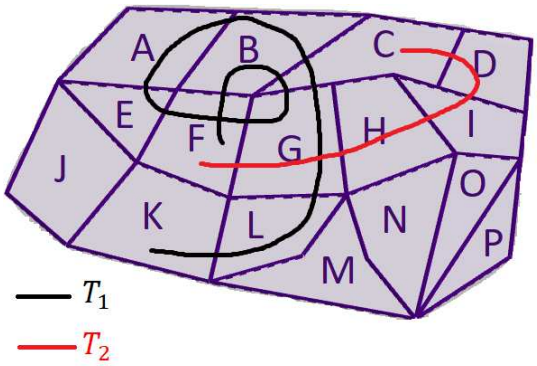
Napomenimo da zapis ($\triangle$) dolazi iz teorije formalnih jezika. Pravila u formalnim jezicima imaju oblik $\alpha \rightarrow \beta$ što znači da se α može zamijeniti s β, a | označava logički operator "ili". Niz predikata u $\mathcal{S}$ definiran je rekurzivno sa $\mathcal{S}.\mathcal{S}$ tako da se operator "." pojavljuje između svaka dva prostorno-vremenska predikata $\mathcal{P}$ u $\mathcal{S}$. U FlexTrack upitnom jeziku svaki je prostorni-vremenski predikat $\mathcal{P}_i$ definiran trojkom $\mathcal{P}_i = \langle op_i, \mathcal{R}_i, [int_i] \rangle$. $\mathcal{R}_i$ odgovara unaprijed definiranom prostornom području u Σ ili varijabilnom području u Γ, tj. $\mathcal{R}_i \in \Sigma \cup \Gamma$, operator op_i opisuje topološku vezu koju trajektorija T_{id} i prostorno područje $\mathcal{R}_i$ moraju zadovoljavati tijekom vremenskog intervala int_i. Topološke veze koje se koriste su *jednako, unutar, dodiruje* kao i mnoge druge. Za trajektoriju T_{id} i područje

$\mathcal{R}_i$, operator op_i vraća boolean vrijednost $\mathbb{B} = \{istina, laž\}$ ovisno o tome zadovoljavaju li trajektorija T_{id} i područje $\mathcal{R}_i$ topološku vezu op_i (npr. operator *unutar* će biti *istina* ako se trajektorija T_{id} nalazila unutar područja $\mathcal{R}_i$ za vrijeme intervala int_i). U nastavku pretpostavljamo da je prostorni operator postavljen na *unutar* te je izostavljen iz upita. Unaprijed definirano područje $\mathcal{R}_i \in \Sigma$ korisnik eksplicitno navodi u upitu, dok varijabilno područje predstavlja proizvoljno područje i označava se malim slovima ispred kojeg stoji simbol "@", npr. @x. Varijabilno područje definirano je koristeći simbole iz Γ, gdje je $\Gamma = \{@a, @b, @c, \dots\}$. Ako drugačije nije navedeno varijabla uzima jednu vrijednost (instancu) iz Γ (npr. $@b = A$). Također, općenito se mogu specificirati moguće vrijednosti varijabilnih područja kao podskup od Γ (npr. "sve gradske četvrti s nogometnim terenom"). Napomenimo da se ista područje $@z$ može pojaviti u nekoliko različitih predikata u obrascu $\mathcal{S}$. Svugdje gdje se pojavljulje odnosi se na isto područje. Ovo je jako korisno za određivanje kompleksnih upita koji uključuju posjećivanje istog područja više puta. Pogledajmo idući primjer: $\mathcal{S} = \{@z.?^*.A.@z\}$. Ovaj obrazac $\mathcal{S}$ pronalazi trajektorije koje kreću iz nekog područja označenog varijablom "$@z$", zatim u nekom trenutku prolaze područjem A i odmah nakon vraćaju se u područje iz kojeg su krenule. Primijetimo da se za naše potrebe, zamjenski znak "?" također smatra varijablom. On se odnosi na bilo koje područje, a ako se pojavljuje više puta ne odnosi se nužno na isto područje. Predikat $\mathcal{P}_i$ može sadržavati eksplicitno vremensko ograničenje int_i u formi intervala, što znači da prostorna veza op_i između trajektorije i područja $\mathcal{R}_i$ mora biti ispunjena u točno specificiranom intervalu int_i. Ako vremensko ograničenje nedostaje, tada pretpostavljamo da prostorna veza može biti ispunjena u bilo kojem trenutku za vrijeme životnog vijeka trajektorije.

Sustav za izvrednjavanje upitnih obrazaca

Radi jednostavnosti pretpostavljamo da je prostor podijeljen na dvodimenzionalna nepreklapajuća područja, kao što možemo vidjeti na slici 3.2. Za učinkovito izvrednjavanje upitnih obrazaca uvest ćemo dvije strukture indeksa u obliku uređenih listi, koje su spremljene uz neobrađene podatke o trajektoriji. Za svaku trajektoriju definiramo *listu od trajektorije* koja sadrži sva područja kroz koje ona prolazi, a za svako područje definiramo *listu od područja* koja sadrži sve trajektorije koje kroz to područje prolaze. *Lista od područja A*, $\mathcal{L}_A$ ponaša se kao invertirani indeks koji sadrži sve trajektorije koje su prošle područjem A. Svaki zapis u $\mathcal{L}_A$ sadrži identifikator trajektorije T_{id}, vremenski interval ($vrijeme\ ulaza, vrijeme\ izlaza$) za vrijeme kojeg je objekt u pokretu bio unutar područja A i pokazivač na *listu od trajektorije* T_{id}. Ako trajektorija prolazi kroz područje A više puta u različitim vremenskim intervalima, bilježi se zapis o svakoj posjeti. Zapis u *listi od trajektorije* T_{id} sadrži područje i vremenski interval ($vrijeme\ ulaska, vrijeme\ izlaska$) tijekom kojeg je trajektorija T_{id} posjetila to područje. Na primjer, na slici 3.2 *lista od područja G* bila bi $\{T_1(5,9); T_1(19,22); T_2(15,19), \dots\}$. *Lista od trajektorije* T_1 bila bi

$\{K(1,3); L(3,5); G(5,9); C(9,10); B(10,13); A(13,16); E(16,17); F(17,19); G(19,22); C(22,23); B(23,26); F(26,28)\}$, a od trajektorije T_2 : $\{C(1,5); D(5,7); I(7,11); H(11,15); G(15,19); F(19,23)\}$



Slika 3.2: Podijeljena prostorna domena

Obzirom na dostupne strukture indeksa za izvrednjavanje upitnih obrazaca predlažemo dvije različite strategije. *Index Join Pattern (IJP)* temelji se na operaciji spajanja koja se izvodi preko *liste od područja* koje odgovaraju svakom fiksnom predikatu u obrascu $\mathcal{S}$. *Dynamic Programming Pattern (DPP)* izvodi se usklađivanjem podniza između upitnog obrasca $\mathcal{S}$ i aproksimacija trajektorija pohranjenih kao *liste od trajektorija*. Oba algoritma koriste dvije iste strukture indeksiranja za odbacivanje, ali na različite načine: *IJP* koristi *liste od područja* za odbacivanje i *liste od trajektorija* za spajanje varijabli, a *DPP* uglavnom koristi *liste od trajektorija* za podudaranje podniza i izvodi odbacivanje na temelju presjeka na *listama od područja*.

Index-Join Pattern Algorithm (IJP)

Radi jednostavnosti pretpostavimo da obrazac $\mathcal{S}$ nema eksplicitnih vremenskih ograničenja, te neka $\mathcal{S}$ sadrži n predikata. Od tih n predikata neka je $\mathcal{S}_f$ skup od f fiksnih predikata, a $\mathcal{S}_v$ skup od v varijabilnih predikata ($n = f + v$). Evaluacija od $\mathcal{S}$ se s IJP algoritmom može podijeliti na dva koraka: (i) evaluacija fiksnih predikata i (ii) evaluacija varijabilnih predikata.

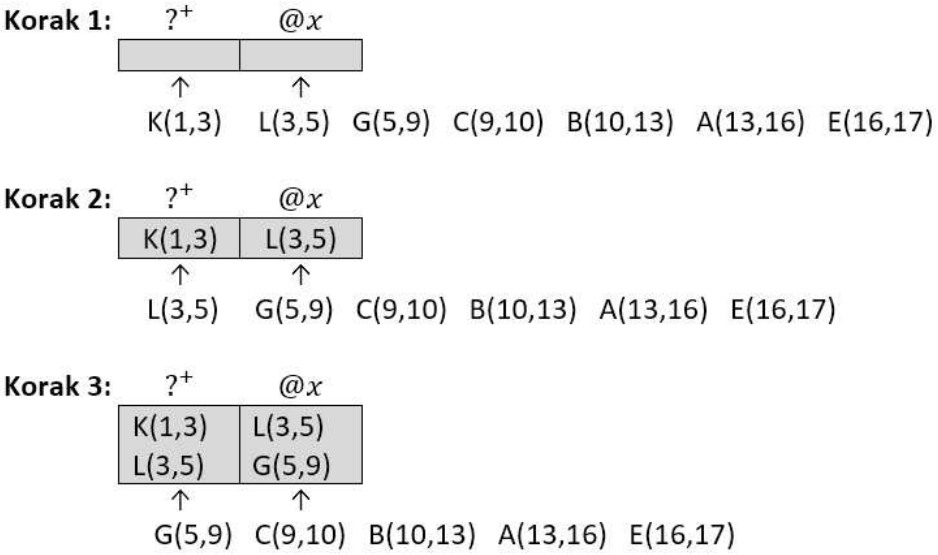
(i) Evaluacija fiksnih predikata: U ovom koraku algoritam procjenjuje skup $\mathcal{S}_f$ koristeći indeks *liste od područja* za brzo odbacivanje trajektorija koje ne odgovaraju obrascu. Svi fiksni predikati u $\mathcal{S}_f$ mogu se evaluirati koristeći operaciju sličnu spajanju između njihovih *lista od područja* $\mathcal{L}_i, i \in 1, \ldots, n$. Zapisi s ovih f popisa dohvaćaju se u sortiranom redoslijedu po T_{id} i zatim spajaju svojim identifikatorim, T_{id}. Zapis su odbačeni koristeći identifikatore trajektorija i vremenske intervale (*vrijeme ulaska*, *vrijeme izlaska*). U svakom popisu $\mathcal{L}_i$ čuvamo pokazivač p_i koji pokazuje na zapis koji se trenutno razmatra za

spajanje. Ako se u obrascu $\mathcal{S}$ neko područje pojavljuje dva puta koristi se dupli pokazivač za to područje.

(ii) Evaluacija varijabilnih predikata: Drugi korak *IJP* algoritma evaluira varijabilne predikate iz $\mathcal{S}_v$, preko skupa kandidata trajektorija U generiranog u prvom koraku. Za fiksni predikat njegova odgovarajuća *lista od područja* sadrži sve trajektorije koje prolaze kroz njega. Međutim, varijabilni predikati se mogu vezati na bilo koje područje, tako da bi morali pogledati sve *liste od područja*, što nije realno ni efikasno. Zbog toga kreiramo nove liste za varijabilne predikate. One se stvaraju u hodu pomoću trajektorija iz U generiranih u prvom koraku. Dakle, za kreiranje lista za varijabilne predikate, za svaki varijabilni predikat $P_j \in \mathcal{S}_v$ izračunavamo sve mogućnosti za varijablu P_j analizirajući *listu od trajektorije* za svaku trajektoriju iz U. Konkretnije, koristimo vremenske intervale trajektorija kako bismo identificirali koji se dijelovi trajektorija mogu dodijeliti ovom konkretnom varijabilnom predikatu.

Koristeći primjer sa slike 3.2 i sljedeći obrazac $\mathcal{S} = \{?^+.@x.?^*.F.?^*.G.?^*.@x.?^*.F\}$, ilustrirajmo *IJP* algoritam. Dakle, obrazac $\mathcal{S}$ traži trajektorije koje najprije posjećuju proizvoljno područje jedan ili više puta ($?^+$), zatim posjećuju neko područje označenu varijablom $@x$, pa nakon posjećivanja nula ili više područja posjećuju područje F, zatim područje G i opet isto područje $@x$ prije vraćanja u F. Za prvi korak algoritma trebamo *liste od područja* F i G. Umjesto dva pokazivača na *list od područja* F, radi jednostavnosti koristimo dvije kopije *liste od područja* F ($\mathcal{L}_{F_1}$ i $\mathcal{L}_{F_2}$). Sada imamo $\mathcal{L}_{F_1} = \mathcal{L}_{F_2} = \{T_1(17,19); T_1(26,27); T_2(19,23), \dots\}$ i $\mathcal{L}_G = \{T_1(5,9); T_1(19,22); T_2(15,19), \dots\}$. Algoritam započinje od prvog zapisi u listi $\mathcal{L}_{F_1}$, preciznije $T_1(17,19)$. Zatim provjera prvi zapis u listi $\mathcal{L}_G$, $T_1(5,9)$. Budući da $T_1(5,9)$ ima interval prije $(17,19)$, u listi $\mathcal{L}_G$ pomičemo se na idući zapis, $T_1(19,22)$. Ova dva pojavljivanja od T_1 podudaraju se s obrascem $F.?^*.G$ u $\mathcal{S}$. Sada moramo provjeriti prolazi li ponovo trajektorija T_1 područjem F. Pogledajmo sada prvi zapis u $\mathcal{L}_{F_2}$, $T_1(17,19)$. Ovaj zapis ne odgovara obrascu jer vremenski interval mora biti nakon intervala $(19,22)$. Pogledajmo stoga sljedeći zapis u $\mathcal{L}_{F_2}$, $T_1(26,27)$. Ovaj zapis odgovara vremenskim ograničenjima i obrascu $\mathcal{S}$. Trajektorija T_1 je kandidat i spremamo je u skup U. Algoritam sada prelazi na drugi zapis u $\mathcal{L}_{F_1}$, $T_1(26,27)$. Lako se vidi da ovaj zapis ne zadovoljava obrazac $\mathcal{S}$. I na kraju, promatramo zapis $T_2(19,23)$ u $\mathcal{L}_{F_1}$. To nas dovodi do zapisa $T_2(15,9)$ u $\mathcal{L}_G$. Međutim, trajektorija T_2 ne može zadovoljiti vremenska ograničenja. Time završava prvi korak algoritma. U drugom koraku analiziramo *liste od trajektorija* za svaku trajektoriju u U. U našem slučaju, jedini kandidat je trajektorija T_1, tj. $T_1 \in U$. Iz prvog koraka znamo da trajektorija T_1 zadovoljava fiksne predikate u sljedećim područjima: $F(17,19), G(19,22), F(26,27)$. Koristeći pokazivače s *lista od područja* iz prethodnog koraka, znamo gdje su odgovarajuća područja na *listi od trajektorije* T_1. Kao rezultat toga, T_1 konceptualno se može podijeliti na tri segmenta S_1, S_2 i S_3. Imamo da je $S_1 = \{K, L, G, C, B, A, E\}$, $S_2 = \{\}$ i $S_3 = \{C, B\}$. Primijetimo da je segment S_2 prazan jer nema područja između F i G. Ovi segmenti koriste se za stvaranje listi za

varijabilne predikate identificiranjem mogućnosti za svaki varijabilni predikat. Budući da svaka mogućnost za varijabilni predikat mora zadovoljavati obrazac, svaki je varijabilni predikat ograničen s dva fiksna predikata koji se pojavljuju prije i poslije njega u obrascu. Svi varijabilni predikati grupiraju se zajedno između dva fiksna predikata. Zatim se za svaku grupu varijabilnih predikata koristi odgovarajući segment trajektorije za generiranje liste za varijabilne predikate za tu grupu. Grupiranje je vrlo korisno te osigurava da varijable u grupi održavaju redoslijed u skladu s obrascem $\mathcal{S}$. Pretpostavimo da se svaka grupa varijabilnih predikata sastoji od w članova. Svaki segment putanje koji utječe na varijable ove grupe "provlači se kroz prozor" veličine w. Prvi element u segment se tada uklanja i prozor se pomiče za jednu poziciju. Ovaj proces nastavlja se sve dok se ne dođe do kraja segmenta. Pogledajmo sada naš primjer. Postoje dvije grupe: prva se sastoji od $?^+$ i $@x$ u tom redoslijedu (tj. $w = 2$), a druga se sastoji od jednog člana $@x$ (tj. $w = 1$). Slika 3.3. prikazuje prva tri koraka u generiranju liste za varijabilne predikate za grupu $?^+$ i $@x$. Ova grupa prolazi kroz segment S_1, te je s desne strane ograničena fiksnim predikatom F. Svaka lista prikazana je ispod odgovarajućeg varijabilnog predikata. Budući da druga grupa s predikatom $@x$ prolazi segmentom S_3, kreira se drugačija lista. Genenirane liste spajaju se na sličan način kao fiksni predikati u prvom koraku. Budući da je lista ispunjena segmentima od iste trajektorije, kriterij spajanja samo provjera je li zadovoljen redoslijed iz obrasca $\mathcal{S}$.



Slika 3.3: Primjer prva tri koraka kako IJP kreira listu za varijabilni predikat

Dynamic Programming Pattern Algorithm (DPP)

Kao i *IJP* algoritam, *DPP* algoritam također se može podijeliti na dva koraka: (i) odabir trajektorije i (ii) usklađivanje. Radi lakšeg opisa dan je pseudokod za *DPP* algoritam (Algoritam 1).

Algorithm 1 DPP

Input: Obrazac $\mathcal{S}$ koji se sastoji od predikata $\mathcal{P}_i$
Output: Trajektorije koje zadovoljavaju $\mathcal{S}$

Nekaj je $\overline{T}$ skup trajektorija kandidata iz *lista od trajektorija* s fiksnim predikatima u $\mathcal{S}$
Skup rješenja $\mathcal{A} \leftarrow \emptyset$
for $T' \in |\overline{T}|$ **do** $Kreiraj\_matricu(T', \mathcal{S})$
    if $Abs(M[|\mathcal{S}|][|T'|]) \geq \mathcal{P}_{|\mathcal{S}|}.idx$ **then** $Pretraži(|\mathcal{S}|, |T'|)$

Funkcija: $Kreiraj\_matricu(T, \mathcal{S})$

for $i \leftarrow 0$ to $|\mathcal{S}|$ **do**
    for $j \leftarrow 0$ to $|T|$ **do**
        if $i = 0$ **or** $j = 0$ **then** $M[i][j] \leftarrow 0$
        else
            if $P_i.type$ fiksni predikat **then**
                if $P_i.R = T.R_j$ **then** $M[i][j] \leftarrow (-(Abs(M[i-1][j-1]) + 1))$
                else $M[i][j] \leftarrow Max(Abs(M[i-1][j]), Abs(M[i][j-1]))$
            else
                if $P_i.type = \{?^+, @\}$ **then** $M[i][j] \leftarrow (-(Abs(M[i-1][j-1]) + 1))$
                else
                    if $i = P_i.idx$ **then** $M[i][j] \leftarrow Abs(M[i-1][j])$
                    else $M[i][j] \leftarrow (-(Abs(M[i-1][j-1]) + 1))$

Funkcija: $Pretraži(i, j)$

if $i > 0$ **then**
    for $k \leftarrow j$ to $k \geq P_i.idx$ **downto** 1 **do**
        if $Abs(M[i][k]) \geq P_i.idx$ **then**
            if $M[i][k] \leq 0$ **then**
                if $P_i.type = \{@\}$ **and** $Match[P_i.link] \neq T'.R_k$ **then continue**
                $Match[i] \leftarrow T'.R_k$
                if $P_i.type = \{?^*\}$ **then** $Pretraži(i-1, k)$
                else $Pretraži(i-1, k-1)$
else $\mathcal{A} \leftarrow \mathcal{A} \cup T'.id$

(i) Odabir trajektorije: Koristeći *liste od područja* prvi korak stvara skup kandidata $\overline{T}$ na temelju fiksnih predikata u $\mathcal{S}$. Za svaku *listu od područja* za fiksna područja u $\mathcal{S}$, odabiremo identifikatore T_{id} onih trajektorija koje su posjetile to područje. Skup kandidata $\overline{T}$ dobivamo presijecanjem prikupljenih identifikatora (jedan skup po svakom području). Zapravo, $\overline{T}$ sadrži sve identifikatore trajektorije koje su neovisno o redoslijedu posjetile sva područja u $\mathcal{S}$.

(ii) Usklađivanje: Drugi korak koristi podudaranje obrazaca za eliminiranje trajektorija koje ne odgovaraju redoslijedu predikata u $\mathcal{S}$. U prethodnom koraku redoslijed područja nije određen, zbog toga se u ovom koraku provodi korak verifikacije za svaku trajektoriju $T' \in \overline{T}$ kako bi bio zadovoljen redoslijed u $\mathcal{S}$. Za svaku se trajektoriju $T' \in \overline{T}$ kreira dinamička matrica M (funkcija *Kreiraj_matricu*). Ona omogućuje *DPP* algoritmu da se podudara sa svim pojavljivanjima obrasca $\mathcal{S}$ u T' u odgovarajućem redoslijedu definiranom u $\mathcal{S}$. Matrica M sadrži stupac j za svako područje koje trajektorija T' posjećuje. Redovi i u matrici odgovaraju predikatima $P_i \in \mathcal{S}$. Stoga je dimenzija matrice $|\mathcal{S}| \times |T'|$. Vrijednosti u svakoj ćeliji $M[i][j]$ izračunavaju se na temelju predikata P_i i j-tog elementa u području aproksimacije trajektorije T' označenog s T_j. To je zapravo j-ti element u *popisu trajektorija* za T'.

Matrica M ispunjava se red po red, stupac po stupac. Funkcija *Kreiraj_matricu* u svakom koraku uspoređuju vrijednost predikata P_i i trenutnog područja iz aproksimacije trajektorije T'_j. Vrijednosti u matricu mogu biti u rasponu od $(-|\mathcal{S}|, |\mathcal{S}|)$. Apsolutna vrijednost odgovara duljini najduljeg podudaranja između obrasca $\mathcal{S}$ i trajektorije T' koja je dosad otkrivena. Negativna vrijednost u $M[i][j]$ označava podudaranje između predikata P_i i područja trajektorije R_i, a njegova apsolutna vrijednost je duljina najduljeg podudaranja do sad. Postoji i poseban slučaj u kojem predikat P_i nije obavezan u obrascu $\mathcal{S}$. U ovom slučaju, izračun i daljnje pretraživanje matrice moraju uzeti u obzir slučaj u kojem P_i ne odgovara T'_j. Kako bi se to riješilo, atribut idx pridružuje se svakom predikatu P_i u $\mathcal{S}$. Ovaj atribut sprema poziciju svakog predikata P_i u slučajevima kada se izborni predikat ne podudara ni s jednim T'_j. Atribut idx definiran je sljedećom formulom

$$P_i.idx = \begin{cases} 1 & \textbf{ako} \quad i = 1 \\ P_{i-1}.idx & \textbf{ako} \quad P_i.type = \{?^*, ?^\#\} \\ P_{i-1}.idx + 1 & \textbf{inače} \end{cases}$$

Nakon što je matrica M kreirana, moraju se pretražiti podudaranja. Budući da negativni brojevi označavaju podudaranje, funkcija *Pretraži* traži negativne vrijednosti u matrici M. Pretraživanje se izvodi u obrnutom redoslijedu od generiranja matrice, dakle počevši s ćelijom u desnom donjem kutu. Ako je posljednji zapis $M[|\mathcal{S}|][|T'|]$ ima apsolutnu vrijednost veću od zadnjeg idx u P, tada postoji najmanje jedno podudaranje između $\mathcal{S}$ i T'. Kako nas samo zanima pronalaženje najdužeg i potpunog podudaranja između $\mathcal{S}$ i T', tražimo samo one unose koji imaju vrijednost veću ili jednaku od $P_i.idx$. Ako je vrijed-

nost manja od $P_i.idx$, funkcija *Pretraži* prekida obradu trenutnog retka i. Ponovo koristeći primjer sa slike 3.2 i isti obrazac $\mathcal{S}$ ilustrirajmo kako radi *DPP* algoritam. Koristeći *liste od područja* za fiksne predikate, odnosno područja F i G u obrascu $\mathcal{S}$ dolazimo do zaključka da trajektorije T_1 i T_2 prolaze kroz oba područja i njih spremamo u skup $\overline{T}$. Za svaku trajektoriju konstruiramo matricu koristeći funkciju *Kreiraj_matricu*. Matrica M za trajektoriju T_1 prikazana je u tablici 3.1. Budući da mi želimo pronaći najduže i potpuno

			T_1	K	L	G	C	B	A	E	F	G	C	B	F
			j	1	2	3	4	5	6	7	8	9	10	11	12
$\mathcal{S}$	i	idx	0	0	0	0	0	0	0	0	0	0	0	0	0
$?^+$	1	1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$@x$	2	2	0	0	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
$?^*$	3	2	0	0	-2	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
F	4	3	0	0	0	3	3	3	3	3	-4	4	4	4	-4
$?^*$	5	3	0	0	0	-3	-4	-4	-4	-4	-4	-5	-5	-5	-5
G	6	4	0	0	0	0	4	4	4	4	4	-5	5	5	5
$?^*$	7	4	0	0	0	0	-4	-5	-5	-5	-5	-5	-6	-6	-6
$@x$	8	5	0	0	0	0	0	-5	-6	-6	-6	-6	-6	-7	-7
$?^*$	9	5	0	0	0	0	0	-5	-6	-7	-7	-7	-7	-7	-8
F	10	6	0	0	0	0	0	0	6	7	-8	8	8	8	-8

Tablica 3.1: Matrica M za trajektoriju T_1 i obrazac $\mathcal{S}$

podudaranje između $\mathcal{S}$ i T_1, tražimo samo one unose koji imaju vrijednost veću ili jednaku od $P_i.idx$. Zbog toga dodatno stavljamo da je $M[i][j] = 0$ tamo gdje je $j < P_i.idx$. Kako je $P_{|\mathcal{S}|}.idx = 6$, funkcija *Pretraži* traži one vrijednosti u 10. retku matricu za koje je $Abs(M[10][j]) \geq 6 = P_i.idx$. Ćelija $M[10][12]$ prolazi kroz sve provjere algoritma te se $M[10][12]$ sprema kao podudaranje u $Match[10]$. Sada se poziva funkcija *Pretraži* za matricu $M[9][11]$. Ćelija $M[9][11]$ prolazi sve provjere pa funkciju *Pretraži* zovemo za $M[8][11]$. Kako je P_8 varijabilni predikat ($P_8 = @x$), te je to prvo pojavljivanje tog predikata, on prolazi provjeru (*link* test) i pridružujemo mu područje B. Zatim funkciju *Pretraži* zovemo idućim redoslijedom: $M[7][10]$, $M[6][10]$. Međutim, za $M[6][10]$ nisu zadovoljeni uvjeti algoritma pa zatim funkciju zovemo za $M[6][9]$, $M[5][8]$, $M[4][8]$, $M[3][7]$ i zatim za $M[2][7]$. Za $M[2][7]$ nisu zadovoljeni uvjeti, odnosno ne prolazi *link* test ($M[2][7] \neq M[8][11]$). Također ni $M[2][6]$ ne prolazi *link* test, ali $M[2][5]$ prolazi *link* test jer je varijabli $@x$ pridruženo područje B ($M[2][6] = M[8][11]$). Funkcija *Pretraži* se tada poziva za $M[1][4]$ sve dok j nije 0. I na kraju, pronađe se obrazac $?^+.B.?^*.F.?^*.G.?^*.B.?^*.F$ i dodaje u $\mathcal{A}$. Pozivanjem funkcije za ostale ćelije, dolazimo do ćelije $M[8][10]$ te ne kraju do obrasca $?^+.C.?^*.F.?^*.G.?^*.C.?^*.F$. Ova su jedina dva obrasca pronađena za $\mathcal{S}$ u T_1. Ako funkciju *Pretraži* zovemo za ostale unose, npr. za $M[10][8](-8)$, vidimo da oni ne prolaze

uvjete algoritma, tj. oni se zapravo ne uspiju povezati s drugim predikatima u $\mathcal{S}$. Obrasce koje smo pronašli za $\mathcal{S}$ u trajektoriji T_1 istaknuti su bojama u tablici. Prvi obrazac označen je plavom bojom, drugi sivom, a zelenom bojom označeni su unosi koji su pronađeni za oba obrasca.

Poglavlje 4

Studijski primjer

4.1 Opis podataka

Podaci pomoću kojih je konstruirana baza podataka preuzeti su s Kaggle online platforme. Kaggle je platforma znanstvenika i podatkovnih entuzijasta koja je primarno namijenjena za razmjenu iskustva, kodova i znanja. Kaggle omogućuje korisnicima da pronađu i objavljuju skupove podataka, istražuju i stvaraju modele u podatkovnom okruženju, rad s drugim znanstvenicima te sudjelovanje u natjecanjima.

Odabrani podaci opisuju kretanje 442 taksija u razdoblju od 1. srpnja 2013. do 30. lipnja 2014. godine. Navedeni taksiji voze u gradu Portu u Portugalu. Dobiveni podaci nisu prilagođeni niti obrađeni. Ti podaci nalaze se u train.csv datoteci. Jedan redak (zapis u tablici) odgovara jednoj vožnji te sadrži 9 atributa koji su prikazani u tablici 4.1. U nastavku opisujemo svaki atribut.

- TRIP_ID - Sadrži jedinstveni id za svaku vožnju.
- CALL_TYPE - Opisuje na koji je način inicirana usluga prijevoza. Sadrži jednu od tri moguće vrijednosti: A (vožnja je inicirana od centrale), B (vožnja je zatražena od taksista na određenom taksi stajalištu) ili C (druge mogućnosti, npr. vožnja je zatražena na slučajnom mjestu u ulici, mahanjem rukom i tome slično).
- ORIGIN_CALL - Sadrži jedinstveni identifikator svakog telefonskog broja koji je korišten za traženje vožnje. Zapravo, identificira putnika ako je CALL_TYPE = A, odnosno ako je vožnja inicirana od centrale. U ostalim slučajevima vrijednost je NULL.
- ORIGIN_STAND - Sadrži jedinstveni identifikator taksi stajališta. Odnosno, ako je CALL_TYPE = B predstavlja lokaciju ukrcaja putnika, odnosno početnu lokaciju putovanja. U ostalim slučajevima vrijednost je NULL.

TRIP_ID	CALL_TYPE	ORIGIN_CALL	ORIGIN_STAND	TAXI_ID
1372636858620000000	C	null	null	20000589
TIMESTAMP	DAY_TYPE	MISSING_DATA	POLYLINE	
1372636858	A	False	[[-8.618643,41.141412], [-8.618499,41.141376], [-8.620326,41.14251], [-8.622153,41.143815], [-8.623953,41.144373], [-8.62668,41.144778], [-8.627373,41.144697], [-8.630226,41.14521], [-8.632746,41.14692], [-8.631738,41.148225], [-8.629938,41.150385], [-8.62911,41.151213], [-8.629128,41.15124], [-8.628786,41.152203],[-8.628687,41.152374], [-8.628759,41.152518], [-8.630838,41.15268], [-8.632323,41.153022], [-8.631144,41.154489], [-8.630829,41.154507], [-8.630829,41.154516], [-8.630829,41.154498], [-8.630838,41.154489]]	

Tablica 4.1

- TAXI_ID - Sadrži jedinstveni id taksista koji je obavio vožnju.
- TIMESTAMP - Unix vrijeme mjereno brojem sekundi od 1. siječnja 1970.godine. Predstavlja vremenski početak vožnje.
- DAYTYPE - Opisuje tip dana na koji je započeta vožnja. Sadrži jednu od tri moguće vrijednosti: B (ako je vožnja započeta na praznik), C (ako je vožnja započeta na dan prije dana tipa B) ili A (inače).
- MISSING_DATA - Vrijednost je FALSE ako su lokacije za vožnju zabilježene, a TRUE ako jedna ili više lokacija nedostaje.

- POLYLINE - Sadrži listu GPS koordinata. Svaki par koordinata reprezentiran je s [LONGITUDE, LATITUDE], odnosno geografskom dužinom i širinom. Ovaj popis sadrži jedan par koordinata svakih 15 sekundi vožnje. Prvi par predstavlja početnu lokaciju putovanja, a zadnji odredište, odnosno lokaciju iskrcaja putnika.

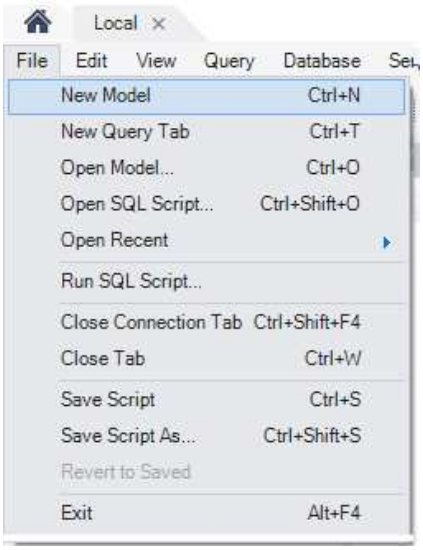
4.2 Kreiranje baze podataka

Pomoću dobivenih podataka kreirat ćemo našu bazu podataka pod nazivom *taxi_tracker*. Baza se sastoji od tri tablice : *taxi*, *trip* i *location*. Tablica *taxi* sadrži atribut *taxi_id* koji predstavlja id taksista. Tablica *trip* sadrži atribute *trip_id, call_type, origin_call, origin_stand, day_type, start_of_trip* i *taxi_id*. Svi atributi osim atributa *start_of_trip* odgovaraju atributima iz datoteke *trip.csv*. Atribut *start_of_trip* označava početak vožnje te će biti u DATETIME formatu. Ovaj podatak dobit ćemo tako da TIMESTAMP atribut iz *train.csv* konvertiramo u DATETIME format. Te na kraju tablica *location* sadrži atribute *time*, *trip_id*, *gps_location* i *neighbourhood*. Ova tablica nam opisuje u koje se vrijeme (*time*), na kojem mjestu (*gps_location*) i u kojem susjedstvu (*neighbourhood*) nalazio taxi koji je vozio vožnju pod brojem *trip_id*.

Za lakše izrađivanje tablica i snalaženje u bazi najprije izradimo relacijski model. Primarni ključ je podcrtan ravnom, a strani ključ isprekidanom linijom. Relacijski model:

- taxi $\left(\underline{\text{taxi\_id}}\right)$
- trip $\left(\underline{\text{trip\_id}}, \text{call\_type}, \text{origin\_call}, \text{origin\_stand}, \text{day\_type}, \text{start\_of\_trip}, \underline{\text{taxi\_id}}\right)$
- location $\left(\underline{\text{time}}, \underline{\text{trip\_id}}, \text{gps\_location}, \text{neighbourhood}\right)$

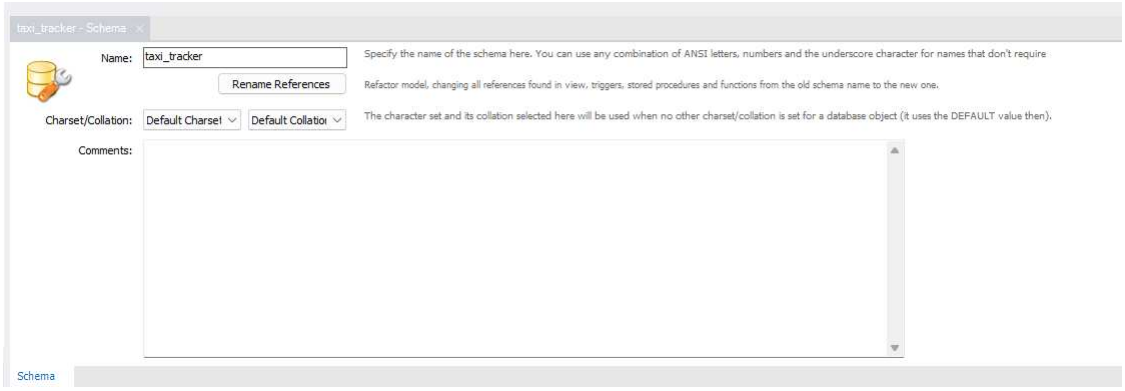
MySQL Workbench



Slika 4.1: Kreiranje baze

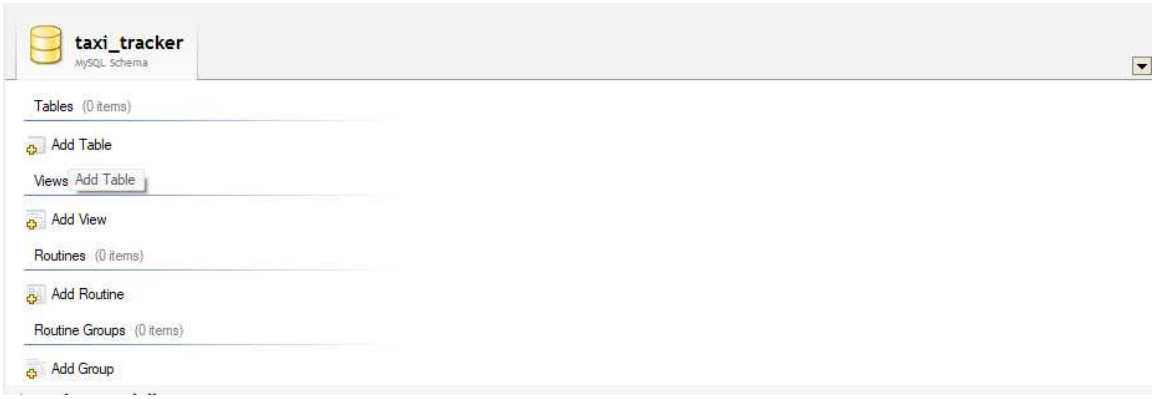
Pomoću MySQL Workbench-a izrađena je baza podataka *taxi_tracker*. Alat je prilično jednostavan za korištenje i snalaženje. Prilikom kreiranja tablica unutar baze važno je slijediti redoslijed. Prvo je potrebno kreirati tablicu *taxi*, zatim tablicu *trip* jer ona sadrži strani ključ koji se referencira na prvu tablicu te na kraju tablicu *location* jer ona sadrži strani ključ koji se referencira na drugu tablicu. Ako tablice ne kreiramo navedenim redoslijedom SQL javlja grešku. Za kreiranje nove baze podataka, na početnoj stranici odaberemo File → New Model (slika 4.1).

U Physical Schemas alatnoj traci kliknemo na + gumb za dodavanje nove scheme. Zadani naziv scheme je new_schema1, ali mi lagano možemo promijeniti ime scheme promjenom polja Name u *taxi_tracker* (slika 4.2). Kada smo kreirali bazu potrebno je kreirati tablice koje baza sadrži. MySQL Workbench omogućuje brzu i jednostavnu izradu tablica.



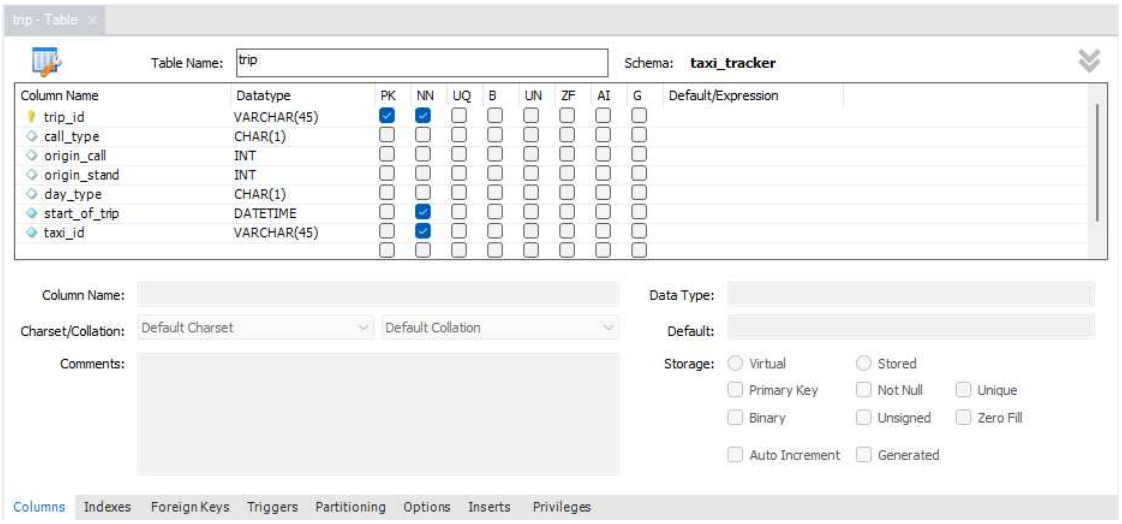
Slika 4.2: Promjena imena

Tablica se kreira tako da dva puta kliknemo na Add Table te se tada otvara prozor u kojem popunjavamo podatke o tablici kao što su ime tablice, imena atributa, tip podatka i sl.

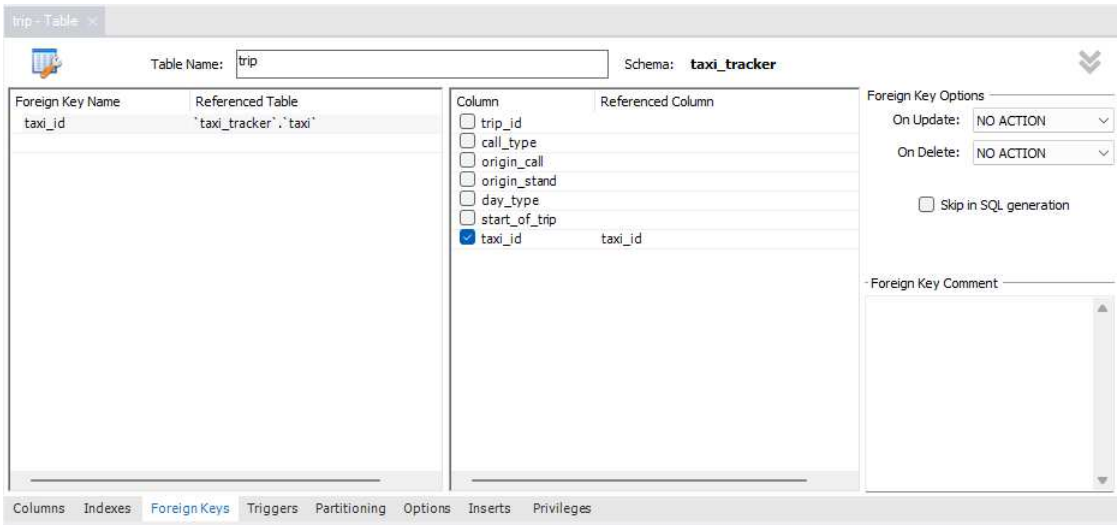


Slika 4.3: Dodavanje tablice

Vrlo je jednostavno odabrati je li atribut primarni ključ (PK) ili ne. Pomoću padajućeg izbornika biramo tip podatka (Datatype). Kako bi dodali vanjske ključeve u tablicu *trip* i *location* u donjoj traci kliknemo na Foreign Keys te ga jednostavno definiramo. Slike 4.4 i 4.5 prikazuju kreiranje tablice *trip* te dodavanje stranog ključa u tu tablicu.



Slika 4.4: Dodavanje tablice trip

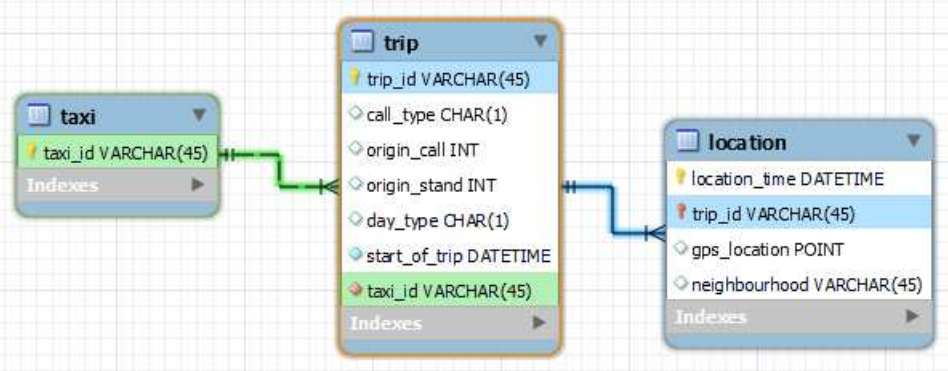


Slika 4.5: Dodavanje stranog ključa

EER (engl.*Enhanced Entity-Relationship*) dijagrami bitan su dio sučelja za modeliranje u MySQL Workbenchu. Oni nam omogućuju vizualni prikaz odnosa između tablica. Na slici 4.6 prikazan je naš EER dijagram. Možemo vidjeti da su primarni ključ (žuti ključić) i strani ključ (crveni ključić) označeni te su odnosi između tablica označeni crtama. Kako bi kreirali EER dijagram kliknemo na Model → Create Diagram from Catalog Objects. Nakon navedenih koraka otvara nam se prozor u kojem je prikazan EER dijagram. Uočimo da smo svaki par koordinata, tj. lokaciju spremili u POINT tip podatka. POINT je jedan

od tipova koji postoje za geometrijske podatke. U dvodimenzionalnom prostoru to je točka definirana X i Y koordinatama, a u našem slučaju geografskom dužinom i širinom.

Kako bi mogli koristiti bazu i povezivati se na nju, potrebno je sinkronizirati model s MySQL serverom, odnosno izvršiti SQL upite koji će kreirati fizičku bazu podataka na MySQL serveru koristeći kreiranu schemu. Kliknemo na Database → Forward Enginer. Nakon što se otvori prozor provjerimo jesu li svi parametri za povezivanje na bazu ispravni te kliknemo na Next → Next. U idućem prozoru provjerimo je li označeno *Export MySQL Table Objects*, pa opet Next → Next → Close. Sada ja baza *taxi_tracker* prisutna na MySQL serveru.



Slika 4.6: EER dijagram

4.3 Povezivanje na bazu i obrada podataka

Nakon što smo kreirali bazu podataka moramo ju ispuniti podacima. Tablicu ćemo ispuniti podacima koristeći Python. Kako bi se povezali na bazu podatka *taxi_tracker*, najprije moramo uspostaviti vezu s MySQL serverom. Sljedeći kod prikazuje kako se povezati na *taxi_tracker* bazu. Metodi `connect` pristupamo preko klase `connector` te joj prosljeđujemo naše parametre za povezivanje. U 11. liniji koda kreirali smo *cursor* objekt te ga nazvali `cursor`. To je objekt koji igra važnu ulogu u izvršavanju upita i dohvaćanju zapisa iz baze podatka. Linije 20-26 u ovom trenutku možemo zanemariti, one nisu potrebne za spajanje na bazu, ali su potrebne za daljnje izvršavanje koda. U nastavku teksta objašnjeno je što rade funkcije `fill_database()`, `mark()` i `find_trips()`.

```
1  import mysql.connector
2  from mysql.connector import Error
3  import datetime
4  import pandas
```

```
5  import folium
6  from geopy.geocoders import Nominatim
7  import random
8  try:
9      connection = mysql.connector.connect(
10                     host = 'localhost',
11                     database = 'taxi_tracker',
12                     user = 'root',
13                     password = 'password')
14
15     if connection.is_connected():
16         cursor = connection.cursor()
17         print("You're connected to database: ",
18             cursor.fetchone())
19
20         fill_database(cursor)
21
22         all_colors=('red', 'blue', 'green', 'purple', 'orange', '
               darkred', 'lightred', 'beige', 'darkblue', 'darkgreen', '
               cadetblue', 'darkpurple', 'white', 'pink', 'lightblue', '
               lightgreen', 'gray', 'black', 'lightgray')
23
24         first_map = mark(cursor,'1372637303620000596', all_colors)
25         first_map.save("Trip.html")
26         find_trips(cursor, all_colors)
27
28 except Error as e:
29     print("Error while connecting to MySQL", e)
30 finally:
31     if connection.is_connected():
32         cursor.close()
33         connection.close()
34         print("MySQL connection is closed")
```

Nakon što smo se uspješno spojili na našu bazu potrebno je učitati *train.csv* datoteku te podatke prilagoditi za SQL bazu podatka. Sljedeći kod prikazuje kako uspješno čitati csv datoteku. Potrebno je uključiti Python paket `pandas` pomoću kojeg učitavamo csv datoteku. Metoda `read_csv()` čita csv datoteku te je sprema u oblike DataFrame-a. DataFrame je struktura podataka koja organizira podatke u tablicu. U petoj liniji koda zapravo brišemo prazne zapise, odnosno prazan par koordinata.

```
1  def read_data():
2
3      taxi_data = pandas.read_csv("C:/Users/monik/train.csv",
           delimiter = ',', nrows = 50)
4      taxi_data = taxi_data[taxi_data.POLYLINE != '[]']
```

```
5       taxi_data['POLYLINE'] = taxi_data.POLYLINE.apply(lambda x:eval(x
            .split()[0]))
6
7       return taxi_data
```

Kada su podaci spremljeni u DataFrame prilično ih je jednostavno dohvatiti. Sljedeći kod prikazuje kako smo dohvatili podatke iz DataFrame-a. DATETIME tip podatka u MySQL-u je tip koji se koristi za spremanje vrijednosti koje sadrže i datum i vrijeme. MySQL prikazuje te podatku u 'YYYY-MM-DD hh:mm:ss' formatu. Python-ov *datetime* modul sadrži klase za rad s datumom i vremenom. Koristeći metodu *timestamp()* iz navedong modula konvertiramo timestamp u datetime format.

```
1  def fill_database(cursor):
2          taxi_data = read_data()
3
4          for ind in taxi_data.index:
5              trip_id = taxi_data['TRIP_ID'][ind]
6              call_type = taxi_data['CALL_TYPE'][ind]
7              origin_call = taxi_data['ORIGIN_CALL'][ind]
8              origin_stand = taxi_data['ORIGIN_STAND'][ind]
9              taxi_id = taxi_data['TAXI_ID'][ind]
10             day_type = taxi_data['DAY_TYPE'][ind]
11             start_of_trip = datetime.datetime.fromtimestamp(
12                 taxi_data['TIMESTAMP'][ind])
13             locations = taxi_data['POLYLINE'][ind]
14
15             insert_data(cursor, locations, trip_id,
16                 call_type, origin_call, origin_stand,
17                 taxi_id, day_type, start_of_trip)
```

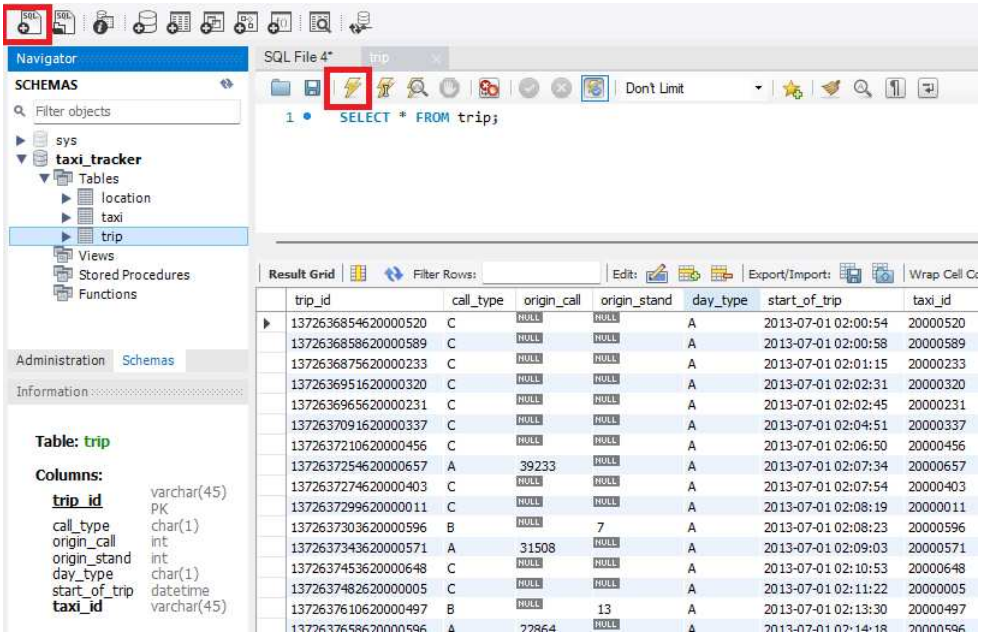
Možemo vidjeti da metoda `fill_database()` koristi metodu `insert_data()` koja nam zapravo omogućuje upis podataka u našu bazu. INSERT INTO izraz koristi se za upisivanje novih zapisa u tablice. `execute()` metoda izvršava upit nad bazom. Linije 6-8 zapisuju podatke u tablicu *taxi*, a linije 14-40 zapisuju podatke u tablicu *trip* ovisno o vrijednosti `call_type` varijable. Ako varijabla `call_type` ima vrijednost A, to znači da varijabla `origin_call` ima vrijednost različitu od null. Slično, ako `call_type` ima vrijednost B, tada je vrijednost od `origin_stand` različita od null. Zadnjih 12 linija, odnosno linije 43-54 zapisuju podatke u tablicu *location*. Koristeći petlju prolazimo kroz sve parove koordinata za jednu vožnju. U varijablu `location_time` spremamo vrijeme koje nam govori u kojem se trenutku vozilo nalazilo na određenoj lokaciji. Kako bi odredili u kojem se susjedstvu nalazio taksi, potrebno je uključiti modul `geopy`. To je Pythonov paket koji sadrži metoda za povezivanje na nekoliko popularnih web servisa za geokodiranje. Jedan od njih je Nominatim. On ima vlastitu klasu u `geopy.geocoders` koja apstrahira

API (eng. *Application Programming Interface*) usluge. Najprije je potrebno inicijalizirati Nominatim API (linija 42), te zatim metodi `reverse()` poslati geografsku širinu i dužina (linija 48). Metoda `reverse()` vraća adresu lokacije u obliku liste. U liniji 49 koristeći `raw()` funkciju raščlanjujemo listu u rječnik te u liniji 50 dolazimo do susjedstva korištenjem `get()` metode.

```
1  def insert_data(cursor, locations, trip_id, call_type,
2              origin_call, origin_stand, taxi_id,
3              day_type, start_of_trip):
4
5      try:
6          insert_taxi_query = """INSERT INTO taxi (taxi_id)
7              VALUES (%s)"""
8          cursor.execute(insert_taxi_query, (str(taxi_id),))
9
10     except Error as e:
11         print("This taxi already exists in table")
12
13     finally:
14         if call_type == 'A':
15             insert_trip_query = """INSERT INTO trip
16                 (trip_id, call_type, origin_call,
17                 day_type,start_of_trip,taxi_id)
18                 VALUES (%s,%s,%s,%s,%s,%s)"""
19             cursor.execute(insert_trip_query, (str(trip_id),
20                 str(call_type), str(origin_call),
21                 str(day_type), str(start_of_trip),
22                 str(taxi_id)))
23         elif call_type == 'B':
24             insert_trip_query = """INSERT INTO trip
25                 (trip_id, call_type, origin_stand,
26                 day_type,start_of_trip,taxi_id)
27                 VALUES (%s,%s,%s,%s,%s,%s)"""
28             cursor.execute(insert_trip_query,
29                 (str(trip_id), str(call_type),
30                 str(origin_stand), str(day_type),
31                 str(start_of_trip),str(taxi_id)))
32
33         else:
34             insert_trip_query = """INSERT INTO trip
35                 (trip_id, call_type, day_type,
36                 start_of_trip,taxi_id)
37                 VALUES (%s,%s,%s,%s,%s)"""
38             cursor.execute(insert_trip_query, (str(trip_id),
39                 str(call_type), str(day_type),
40                 str(start_of_trip),str(taxi_id)))
41
```

```
42          geolocator = Nominatim(user_agent='test')
43          for i in range(len(locations)):
44              location_time = start_of_trip + datetime.timedelta(
                    seconds=i*15)
45              position = locations[i]
46              longitude = position[0]
47              lattitude = position[1]
48              location = geolocator.reverse(str(lattitude)+","+str(
                    longitude),language='en')
49              address = location.raw['address']
50              neighbourhood = address.get('neighbourhood','')
51              insert_query_location = """INSERT INTO location (
                    location_time, trip_id, gps_location, neighbourhood)
                    VALUES (%s,%s,Point(%s, %s),%s)"""
52              cursor.execute(insert_query_location, (str(location_time
                    ),str(trip_id), str(longitude),str(lattitude),str(
                    neighbourhood)))
```

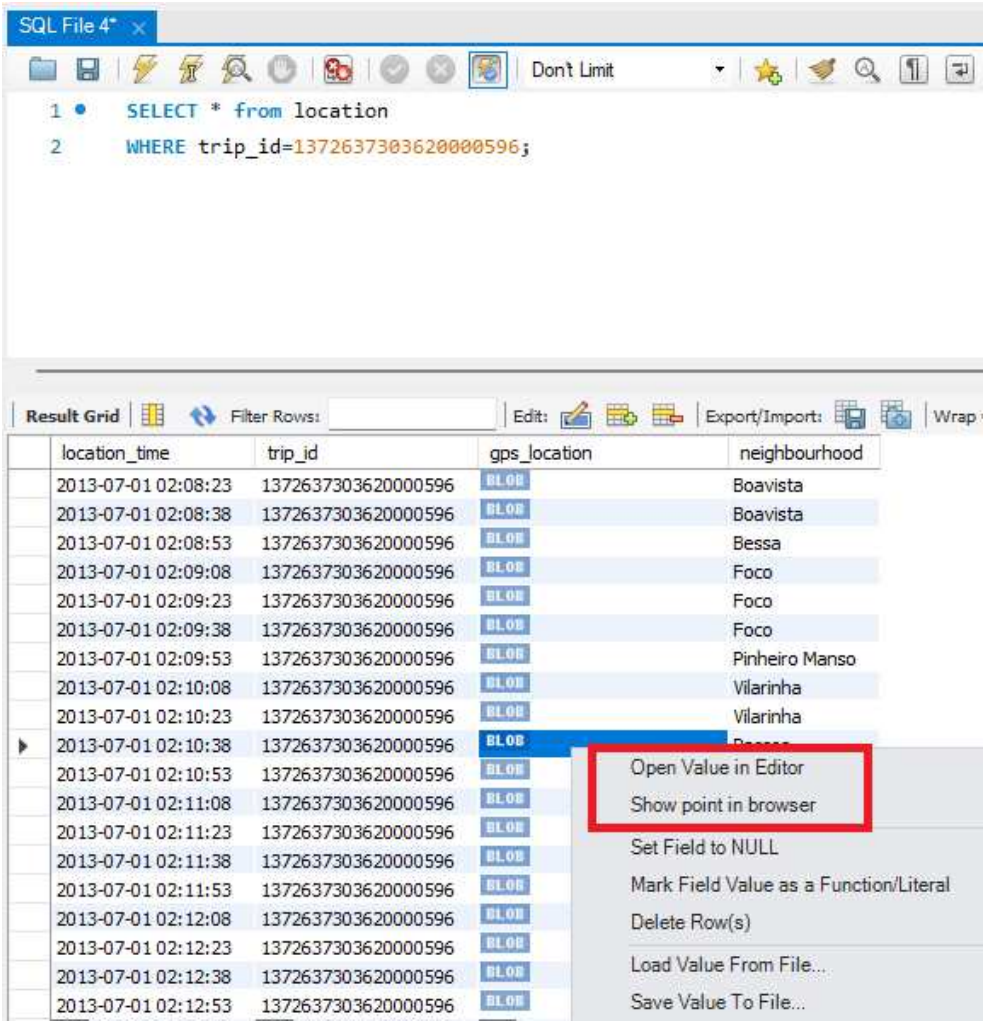
Nakon što smo unijeli podatke u bazu možemo jednostavnim *select* upitima provjeriti jesu li podaci ispravno uneseni u naše tablice. Slike 4.7 i 4.8 prikazuju primjer izvršavanja *select* upita te rezultate.



Slika 4.7: Svi zapisi iz tablice trip

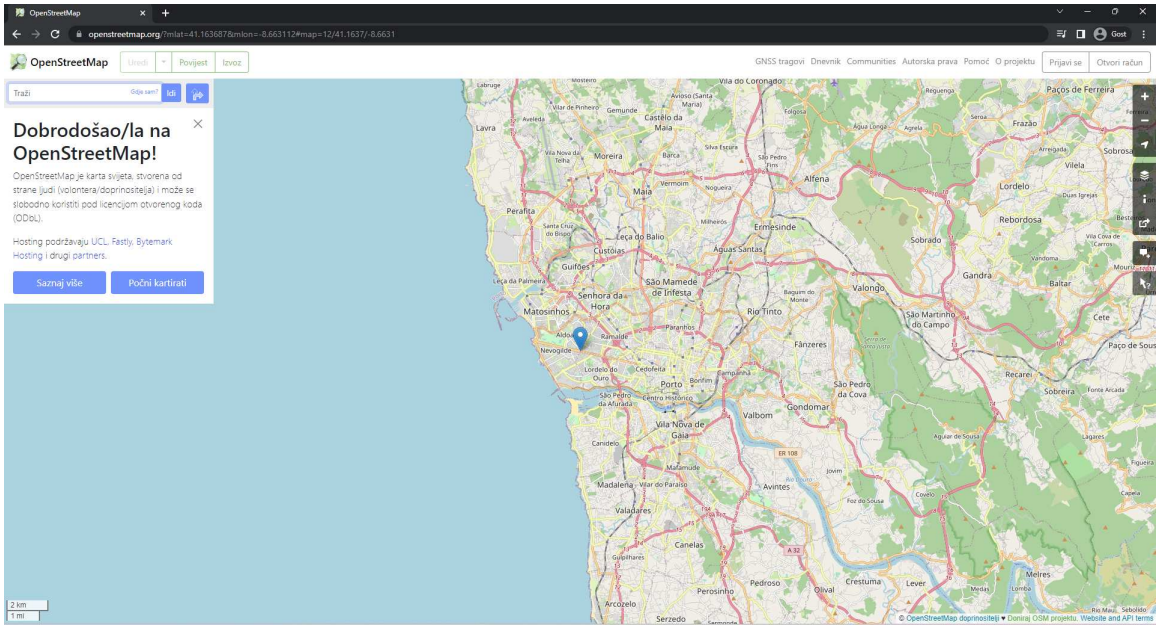
Kako bi u MySQL Workbench-u izvršili SQL upite, najprije se moramo povezati na bazu. Kliknemo na Database → Connect to Database. Zatim na alatnoj traci kliknemo na

gumb za kreiranje novog SQL upita (označeno crvenim okvirom na slici 4.7), napišemo naš upit te kliknemo na gumb za izvršavanje upita. Rezultati su prikazani u prozoru ispod napisanog upita. Na slici 4.8 prikazane su sve lokacije koje je taksi prošao vozeći vožnju pod brojem *trip_id=1372637303620000596*. Možemo vidjeti da su lokacije spremljene binarnom (BLOB) formatu. Ako nas zanima prava vrijednost, desnim klikom na BLOB odabiremo *Open Value in Editor* te se otvara novi prozor koji je prilagođen za rad s binarnim podacima. Na slici 4.9 možemo vidjeti spomenuti prozor u kojem je vidljiva čitljiva vrijednost polja *gps_location*. Također, desnim klikom na BLOB možemo odabrati *Show point in browser*. Ova naredba prikazuje lokaciju u pretraživaču koristeći openstreetmap.org. Rezultat te naredbe vidmo na slici 4.10.



Slika 4.8: Lokacije za vožnju broj 1372637303620000596

Slika 4.9: Open Value in Editor



Slika 4.10: Show point in browser

4.4 Vizualizacija podataka kroz aplikaciju

Kako radimo s podacima koji reprezentiraju vožnju taksija, bilo bi korisno sve lokacije koje su poznate za vožnju prikazati na karti te time dobiti vizualni prikaz kretanja taksija, odnosno vizualni prikaz jedne vožnje. `folium` je vrlo koristan i snažan Python paket koji omogućuje kreiranje nekoliko tipova Leaflet karti. Leaflet je *open-source* JavaScript biblioteka koja olakšava razvoj interaktivnih karata, ali je dizajnirana za korištenje putem JavaScripta. Dakle, `folium` nam omogućuje markiranje lokacija na karti, te tu kartu sprema u zasebnoj HTML datoteci. Sljedeći kod prikazuje implementaciju funkcije `mark()`. Linije 2-6 izvršavaju upit nad bazom. Upit dohvaća sve lokacije za vožnju pod brojem `trip_id=1372636858620000589`. Funkcije `ST_X()` i `ST_Y()` vraćaju x i y koordinate točaka. U našem slučaju vraćaju geografsku dužinu i širinu lokacija. Metoda `fetchall()` dohvaća sve retke iz skupa rezultata upita i vraća listu n-torki, u našem slučaju trojki.

```
1  def mark(cursor, id, all_colors):
2      sql_query = """SELECT ST_X(gps_location), ST_Y(gps_location),
           location_time
3      FROM location
4      WHERE trip_id=%s"""
5      cursor.execute(sql_query,(id,))
6      record = cursor.fetchall()
7
8      markers = []
9      time_moments = []
10     number_of_records = cursor.rowcount
11     latitude_mean = 0
12     longitude_mean = 0
13
14     for row in record:
15         longitude = row[0]
16         lattitude = row[1]
17         time_moments.append(row[2])
18         latitude_mean += lattitude
19         longitude_mean += longitude
20         marker = (lattitude,longitude)
21         markers.append(marker)
22
23     center = [latitude_mean/number_of_records, longitude_mean/
           number_of_records]
24     m = folium.Map(location = center,zoom_start = 14, control_scale
           = True)
25     new_color = random.choice(all_colors)
26     m = add_markers(markers, m, time_moments, new_color)
```

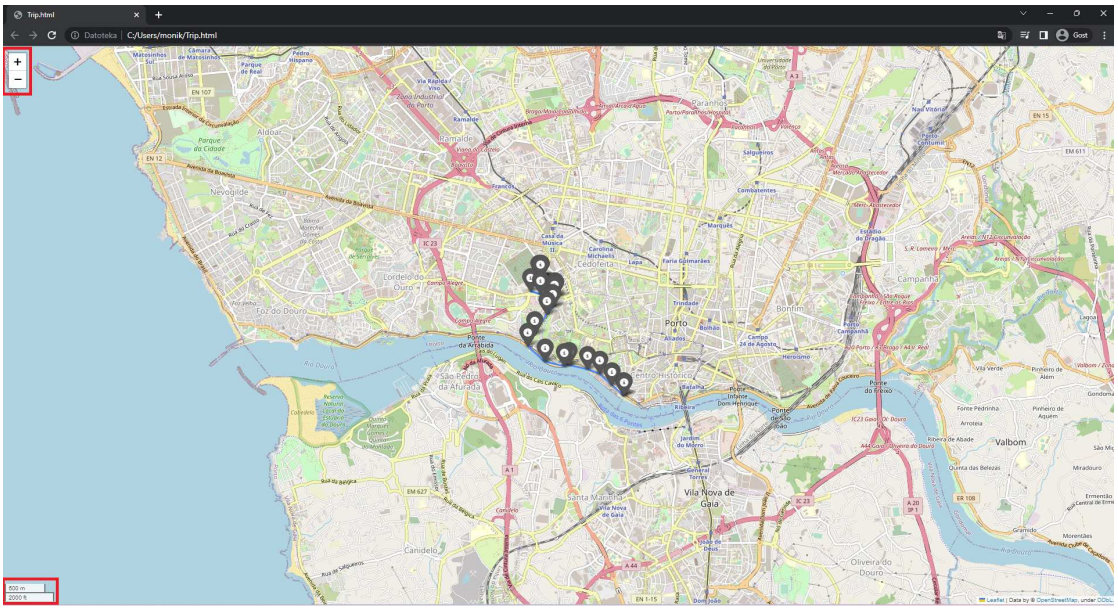
```
27      folium.PolyLine(markers, color = new_color).add_to(m)
28      return m
```

Kako bi najprije prikazali kartu koristeći `folium`, moramo pozvati metodu `Map()`, linija 24. Unutar te metode možemo definirati niz parametara. Neki od njih su `location` (pozicija koja odgovara središtu karte), `zoom_start` (početni nivo zumiranja karte) te `control_scale` (hoće li mjerilo karte biti prikazano). Kao središte naše karte (`center`) koristiti ćemo srednju geografsku širinu i dužinu naših podataka. Lokacije koje želimo markirati, spremili smo u listu `markers`. Kako bi dodali markere na našu kartu implementirali smo funkciju `add_markers()` čiji kod vidimo u nastavku.

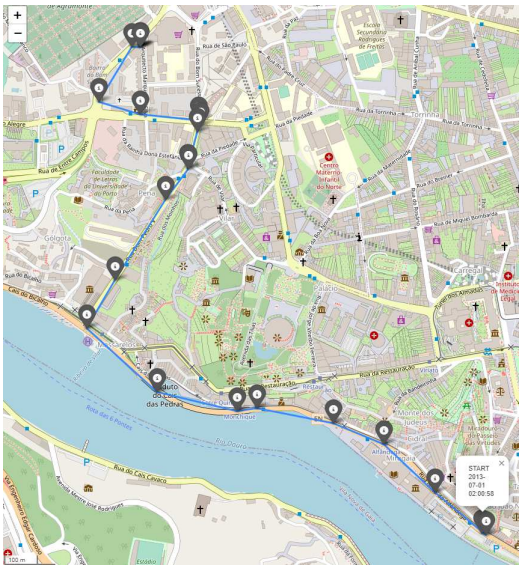
```
1  def add_markers(markers, map_obj,time_moments, new_color):
2      for i in range(0,len(markers)):
3          if i == 0:
4              folium.Marker(markers[i], icon = folium.Icon(color =
                   new_color), popup = 'START ' + str(time_moments[i])).
                   add_to(map_obj)
5          elif i == (len(markers) - 1):
6              folium.Marker(markers[i], icon = folium.Icon(color =
                   new_color), popup = 'END ' + str(time_moments[i].time
                   ())).add_to(map_obj)
7          else:
8              folium.Marker(markers[i], icon = folium.Icon(color =
                   new_color), popup = time_moments[i].time()).add_to(
                   map_obj)
9      return map_obj
```

Markere na kartu dodajemo pozivom metode `Marker()`. Metoda `Marker()` prima lokacije. Da bi se markeri pojavili na karti potrebno je još pozvati funkciju `.add_to()` kojoj kao argument proslijedimo kartu. Moguće je promijeniti boju markera postavljanjem `icon` parametra unutar `Marker()` metode na `folium.Icon(color=new_color)` gdje nama `new_color` predstavlja slučajnu boju izabranu iz skupa boja `all_colors`. Također, dodavanjem parametra `popup` dodajemo labele našim markerima. Svakom našem markeru, odnosno lokaciji, u labelu smo zapisali vrijeme u kojem se taksi nalazio na toj lokaciji, a početnoj i završnoj lokaciji još smo dodali 'START', odnosno 'END'. Nakon što smo označili sve lokacije na karti, sljedeći korak je povezivanje tih lokacija kako bi vizualizirali rutu taksija. Metoda koja nam to omoguće je metoda `Polyline()`. Ona kao parametar prima točke, u našem slučaju lokacije, koje kad se povežu stvaraju izlomljenu liniju. Kao i kod metode `Marker()`, potrebno je pozvati funkciju `add_to()` kako bi se izlomljena linija prikazala na karti. Kada bi imali više podataka, tj. kada bi podatke o lokaciji vožnje imali svakih 2, 3 ili 5 sekundi, a ne 15, prikazana ruta bila preciznija. U tom slučaju mogli bismo izračunatu precizniju trenutnu brzinu vozila. Generirani HTML file, u našem slučaju *Trip.html* vidimo na slici 4.11. U lijevom gornjem kutu nalaze se gumbi za zumiranje, a u lijevom donjem kutu označeno je mjerilo karte. Ako nas zanima labela, tj.

oznaka za pojedini marker, jednostavno kliknemo na marker. Nakon što zumiramo nekoliko puta i kliknemo na početnu lokaciju, odnosno na marker koji reprezentira početnu lokaciju vožnje s brojem *trip_id=1372636858620000058* prikazuje se prozorčić, tj. labela u kojoj piše 'START' što nam označava da je to lokacija ukrcaja putnika te vrijeme i datum kada je ta vožnja započeta. Rezultat vidimo na slici 4.12.



Slika 4.11: Trip.html



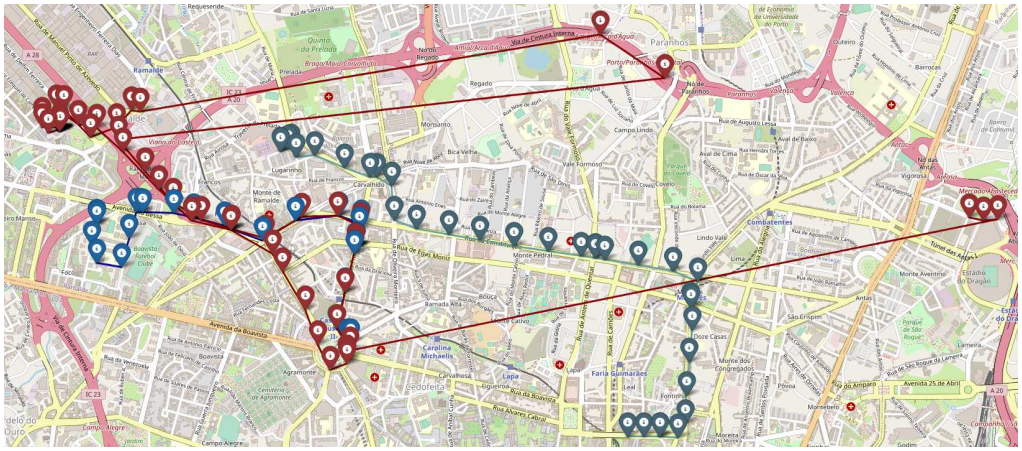
Slika 4.12: Trip.html - prikaz labela

Koristeći znanje iz trećeg poglavlja rada, pronaći ćemo sve vožnje koje zadovoljavaju sljedeći jednostavan obrazac $\mathcal{S}$ = {?*.Ramada Alta.?*.Carvalhido.?*}. Dakle, tražimo one vožnje koje posjećuju proizvoljno područje nula ili više puta (?*), zatim posjećuju susjedstvo Ramada Alta, pa nakon posjećivanja nula ili više područja prolaze kroz susjedstvo Carvalhido te nastavljaju put kroz nula ili više područja. Kako bi pronašli takve trajektorije koristimu varijaciju IJP algoritma. U metodi `find_trips()` najprije dohvaćamo sve vožnje koje prolaze kroz susjedstvo Ramada Alta, te vožnje koje prolaze kroz susjedstvo Carvalhido, linije 2-10. Zatim koristeći petlje prolazimo kroz sve vožnje i tražimo one koje su najprije prošle kroz Ramada Alta, a nakon toga Carvalhido. Vožnje koje zadovoljavaju obrazac $\mathcal{S}$ spremili smo u `tripRAC`. Ponovo koristeći metodu `mark()`, te metodu `add_trip()` prikazujemo rutu za svaku vožnju.

```
1  def find_trips(cursor, all_colors):
2      select_Ramada_Alta = """SELECT trip_id, location_time FROM
           location WHERE neighbourhood='Ramada Alta' ORDER BY
           location_time ASC"""
3
4      cursor.execute(select_Ramada_Alta)
5      recordRA = cursor.fetchall()
6
7      select_Carvalhido = """SELECT trip_id, location_time FROM
           location WHERE neighbourhood='Carvalhido' ORDER BY
           location_time ASC"""
8
9      cursor.execute(select_Carvalhido)
10     recordC = cursor.fetchall()
11
12     tripRAC = set()
13     for ra in recordRA:
14         for c in recordC:
15             if( (ra[0] == c[0]) & (ra[1] < c[1])):
16                 tripRAC.add(ra[0])
17
18     print(tripRAC)
19     i = 0
20     for id in tripRAC:
21         if i==0:
22             m = mark(cursor, id, all_colors)
23             i+=1
24         else:
25             m = add_trip(cursor, id, m, all_colors)
26
27     m.save("TripsRAC.html")
28
29 def add_trip(cursor, id, m, all_colors):
30     sql_query = """SELECT ST_X(gps_location), ST_Y(gps_location),
```

```
            location_time FROM location WHERE trip_id=%s"""
31      cursor.execute(sql_query,(id,))
32      record = cursor.fetchall()
33      markers = []
34      time_moments = []
35
36      for row in record:
37          longitude = row[0]
38          lattitude = row[1]
39          time_moments.append(row[2])
40          marker = (lattitude,longitude)
41          markers.append(marker)
42
43      new_color = random.choice(all_colors)
44      m = add_markers(markers, m, time_moments, new_color)
45      folium.PolyLine(markers, color = new_color).add_to(m)
46      return m
```

Vožnje prikazujemo na jednoj karti te generiranu kartu spremamo u *TripRAC.html*. Rezultat vidimo na slici 4.13. Možemo vidjeti da smo pronašli tri vožnje koje zadovoljavaju obrazac $\mathcal{S}$, i to one s `trip_id` jednakim 1372638361620000154, 1372637610620000497 i 1372638595620000233. Međutim ako bismo iz train.csv datoteke unijeli više zapisa u bazu, a ne samo 50, vjerojatno bi pronašli više vožnji koje zadovoljavaju obrazac $\mathcal{S}$.



Slika 4.13: TripsRAC.html

Zaključak

U ovom diplomskom radu opisane su prostorno-vremenske baze podataka te algoritmi koji se vežu uz njih. Varijacija jednog od algoritama korištena je i u izvedbi studijskog primjera. Podaci koji su korišteni u izradi studijskog primjera preuzeti su s Kaggle online platforme.

Implementacija baze nije bila suviše teška, međutim prikazivanje podataka i isčitavanje informacija iz istih predstavlja nam problem. Prednost ovog pristupa je jednostavnost implementacije i modularnost rješenja. Pri tome mislimo kako je vrlo jednostavno naknadno dodati određene funkcionalnosti i tako proširiti rješenje. Modularnosti doprinosi i programski jezik Python koji omogućuje da se uključivanjem različitih paketa koriste mnoge funkcionalnosti za vizualizaciju, dobivanje informacija i komunikaciju s udaljenim web servisima. Kao još jednu prednost valjalo bi spomenuti kako se izradom vlastitog rješenja ne moramo pouzdati u tuđe proizvode i brinuti o licencima. Međutim, prednost komercijalnih rješenja je podrška, te konstantno ažuriranje koje uključuje popravku *bugova* i dodavanje novih funkcionalnosti. Komercijalna rješenja su uglavnom vrlo dobro dokumentirana zbog čega ih nije problem prilagoditi vlastitim potrebama. Također, zbog velikog broja podataka, dohvaćanje dodatnih informacija o osnovnim podacima zahtjeva velik promet i previše upita na web servis koji te upite može protumačiti kao DoS napad (engl. *Denial of Service*).

Unatoč navedenim nedostacima, prostorno-vremenske baze podataka sve su više zastupljene u današnjim tehnologijama. Kako se korištenjem raznih informacijskih tehnologija generiraju goleme količine podataka koje imaju prostorne i vremenske odrednice, neizbježno je postojanje prostorno-vremenskih podataka. Oni se koriste u različitim područjima poslovanja i znanosti te postoji velika diferencijacija među tim podacima. Zbog toga je potrebno razviti više specijaliziranih softverskih rješenja koja su specijalizirana za rad s određenim tipom podataka.

Na kraju ovog radu možemo zaključiti da ovo područje nije pretjerano istraženo te ima mjesta za razvoj novih tehnologija koje će biti specijalizirane za rad u određenim područjima znanosti i poslovanja.

Bibliografija

[1] Allen J. F., *Maintaining knowledge about temporal intervals*, Communications of the ACM **26** (1983), br. 11, 832–843.

[2] S. Gurram, *Pattern Mining in Spatio-Temporal Databases*, Lambert Academy Publishing, 2018.

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

[4] Autorizirana predavanja FER Napredni modeli i baze podatka, `https://www.fer.unizg.hr/_download/repository/5._Vremenske_baze%5B1%5D.pdf`.

[5] Marcos R Vieira, Petko Bakalov i Vassilis J Tsotras, *Querying trajectories using flexible patterns*, Proceedings of the 13th International Conference on Extending Database Technology, 2010, str. 406–417.

Sažetak

Cilj ovog diplomskog rada bio je predstaviti prostorno-vremenske baze podataka. Prostorno-vremenske baze podataka pojavljuju se u geografskim, meteorološkim i drugim sličnim informacijskim sustavima. Riječ je o bazama gdje su podacima pridružene prostorne i vremenske odrednice. Kroz prvo poglavlje detaljno su opisane prostorne i vremenske baze podataka. Zatim su u sljedećem poglavlju predstavljene prostorno-vremenske baze podataka te primjena istih. U trećem poglavlju predstavljen je FlexTrack sustav koji uvodi novi upitni jezik za postavljanje upita na prostorno-vremenskoj bazi podataka. U konačnici je implementirana prostorno-vremenska baza podataka koja predstavlja kretanje taksi vozila, a uz navedenu bazu podataka izrađeno je i programsko rješenje koje vizualizira jednu taksi vožnju.

Summary

The aim of this thesis is to present spatio-temporal databases. Spatio-temporal databases appear in geographic, meteorological and other similar information systems. These are databases where spatial and temporal determinants are attached to the data. In the first chapter, spatial and temporal databases are described in detail. Then, in the next chapter, spatio-temporal databases and their application are presented. In the third chapter, the FlexTrack system is presented, which introduces a new query language for querying the spatio-temporal database. Finally, a spatio-temporal database representing the movement of taxis was implemented. In addition to the database mentioned, a software solution was created to view a single taxi ride.

Životopis

Rođena sam 23. listopada 1997. godine u Varaždinu. Osnovnu školu završila sam u Trnovcu, a prirodoslovno-matematičku gimnaziju u Varaždinu. Preddiplomski studij matematike na Prirodoslovnom-matematičkom fakultetu u Zagrebu upisala sam 2016., a završila sam 2020. godine. Školovanje sam nastavila na Matematičkom odsjeku upisavši diplomski studij Financijske i poslovne matematike.