UNIVERZA V LJUBLJANI

FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – praktična matematika (VSŠ)

Ada TANKO

PRIKAZ RAZLIČNIH TEHNOLOGIJ PRI GEOGRAFSKIH INFORMACIJSKIH SISTEMIH

Diplomska naloga

Ljubljana 2009

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

PROGRAM DELA

V diplomski nalogi obdelajte določene tehnologije, ki se uporabljajo pri geografskih informacijskih sistemih, kot je na primer objektni model geografskih podatkov. Predstavite način dela z geografskimi podatki s sistemom PostgreSQL z dodatkom PostGIS. Na konkretnem primeru prikažite uporabo teh tehnologij.

mentor

mag. Matija Lokar

POVZETEK VSEBINE

Diplomska naloga zajema predstavitev geografskih podatkov z različnimi tehnologijami, kot so GIS, GIS orodja, PostgreSQL z dodatkom PostGIS in JScript.

Predpostavila sem, da bralec diplomskega dela pozna osnove dela z relacijskimi bazami podatkov in vsaj programski jezik JavaScript ali Java.

Cilj diplomske naloge je torej za vse, ki jih zanima GIS, predstaviti delo z GIS podatki z različnimi tehnologijami.

Diplomska naloga, ki je pred vami, predstavlja na primeru kako geografske podatke za Slovenijo, ki so shranjeni v shape datotekah pretvoriti v relacijsko bazo in obratno. Ker PostgreSQL ne omogoča delo z geografskimi podatki sem predstavila razširitev sistema PostgreSQL z dodatkom PostGIS. Kot programski jezik, ki omogoči povezavo vseh omenjenih tehnologij, sem uporabila JScript. Za grafični prikaz geografskih podatkov za Slovenijo sem uporabila GIS orodje Easy Map Explorer.

Math. Subj. Class. (2000): *68U15, 68P05, 68P15, 68P10, 68N15*

Computing Review Class. System (1998): *H.2.8, H.2.0, H.2.3, H.2.4, H.2.1, D.3.3, C.1.3*

Ključne besede: *geografski informacijski sistem, PostgreSQL, PostGIS, OpenGIS, relacijska baza, geografski podatki, meta podatki, GIS orodja, dimenzionalno razširjen model, geometrijski objektni model, JavaScript ...*

ABSTRACT

This thesis includes a presentation of geographical information by various technologies such as GIS, GIS tools, PostGIS with the add-on for PostgreSQL and JScript.

I presume that the reader knows the basics about databases and at least basics of the programming languages Java or JavaScript.

The goal of this dissertation is to introduce, to all of them who are interested in GIS technologies, how to work with GIS data and different technologies.

This Dissertation will show you in an example, how to transform geographical data for Slovenia, that are stored in shape files, into a relational database and vice-versa. Because PostgreSQL ins't enabling work with geographical data, I introduced the PostGIS extension PostgreSQL. As a programming language, that allows connection of all mentioned technologies, I used Jscript. For the geographical display of geographical data for Slovenia, I used the GIS tool Easy Map Explorer.

Math. Subj. Class. (2000): *68U15, 68P05, 68P15, 68P10, 68N15*

Computing Review Class. System (1998): *H.2.8, H.2.0, H.2.3, H.2.4, H.2.1, D.3.3, C.1.3*

Key words: *geographical information system, PostgreSQL, PostGIS, Open Geospatial Consortium, database, geospatial data, metadata, GIS tools, Dimensionally Extended model, geometry object model, JavaScript ...*

KAZALO

KAZALO SLIK

UVOD

Področje, ki me zanima, je delo z geografskimi informacijski sistemi.
Zato sem se odločila, da bom v diplomski nalogi prikazala različne tehnologije pri geografskih informacijskih sistemih. S tem bom dodatno razširila svoje znanje, ki mi bo v prihodnosti v pomoč pri delu.

V uvodnem poglavju so navedeni pojmi o geografskih informacijskih sistemih ter njihova uporaba. Sledi obsežno poglavje, v katerem je predstavljen PostgreSQL z dodatkom PostGIS. V njem so predstavljene funkcije, ki jih omogoča PostGIS za delo z geografskimi podatki. Nato sledi poglavje, v katerem je predstavljeno kako izmenjujemo geografske podatke med relacijsko bazo in tako imenovanimi shape datotekami, ki jih uporabljajo številna orodja za vizualizacijo geografskih podatkov. Sledi zaključno poglavje, kjer na primeru izdelave karte Slovenije praktično pokažem primer uporabe opisanih tehnologij.

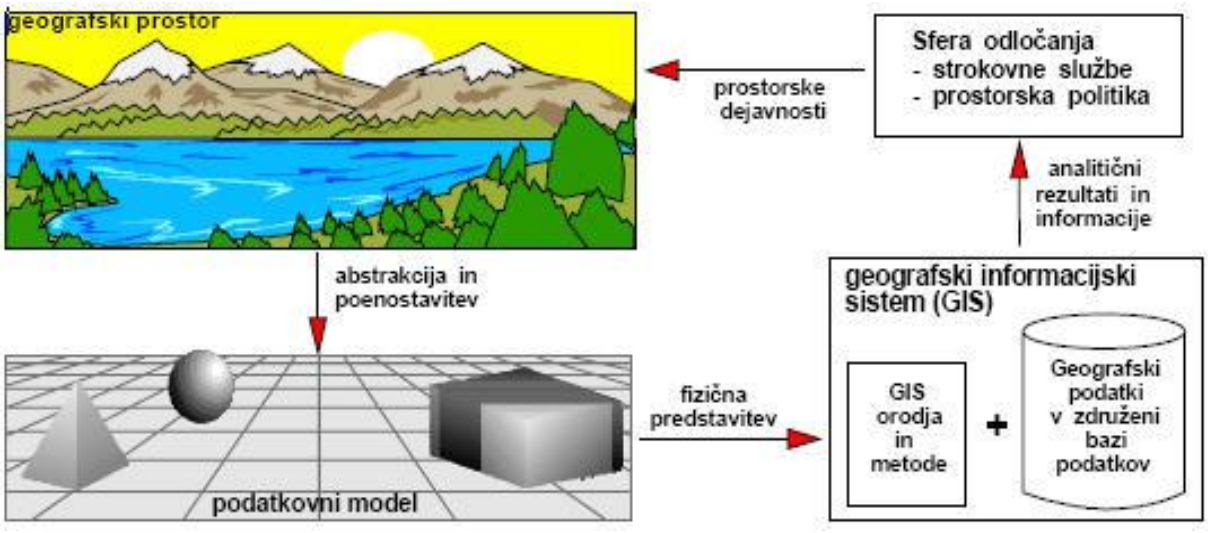
1 NEKAJ POJMOV O GEOGRAFSKIH INFORMACIJSKIH SISTEMIH

1.1 KAJ JE GIS

GIS (geografski informacijski sistem) je sistem za upravljanje z velikimi bazami geografskih podatkov, ki povezujejo lokacijske in opisne podatke o geografskih pojavih. Baze podatkov služijo za zajemanje, shranjevanje, vzdrževanje, obdelavo, analiziranje in predstavitev geografskih podatkov.
Tehnologijo GIS (geografski informacijski sistem) tvorijo orodja, ki so potrebna za uspešno in učinkovito zajemanje, vzdrževanje, obdelave, analize, posredovanje in upravljanje z geografskimi podatki. Shranjeni geografski podatki podajajo lokacijske in opisne značilnosti stvarnih pojavov.

GIS sistem se uporablja predvsem za:

- pridobivanje, zajem in urejanje geografskih podatkov;
- analize geografskih podatkov;
- upravljanje z geografskimi podatki, povezovanje in izmenjavo geografskih podatkov;
- izdelavo digitalnih in topografskih kart;
- 3D modeliranje terena;
- upravljanje GIS sistema in trženje izdelkov.



Slika 1 Grafična predstavitev obdelave in uporabe geografskih podatkov (Tehnologija GIS, Šumrada, 2005, str. 75)

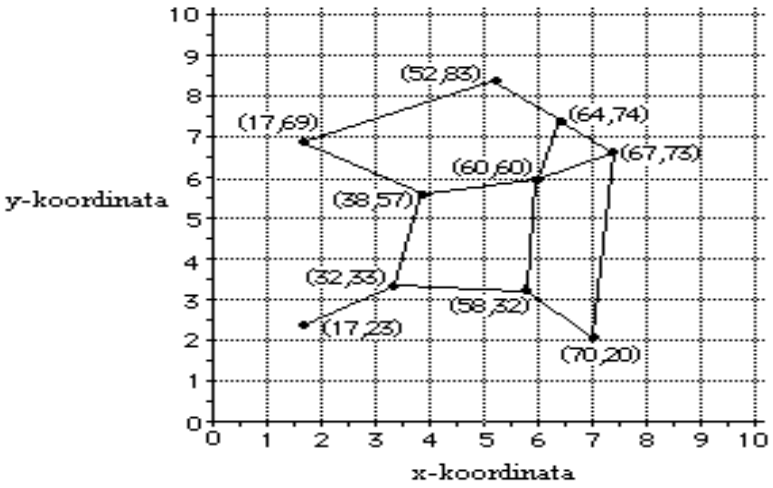
Podatke za GIS ločimo na:

- geografske podatke, ki pomenijo konkretno predstavitev modela stvarnosti. Opisujejo določene lastnosti geografskih objektov (na primer položaj stavbe, širino ceste itd.) in odnose med njimi (ta reka teče pod določenim mostom, določen semafor je ob tej in tej cesti, tista stavba je del tega naselja itd.);
- metapodatke, ki podajajo informacije o sestavi, vsebini, vrednosti, kakovosti, organizaciji, dostopnosti in možni uporabi shranjenih geografskih podatkov. Obsegajo podatke, ki se nanašajo na vsebino, strukturo, kvaliteto, lastništvo, distribucijo, tehnologijo, namen, uporabnost in druge elemente. O metapodatku govorimo takrat, ko le-ta opisuje geografski podatek.

1.2 GEOGRAFSKI IN METAPODATKI

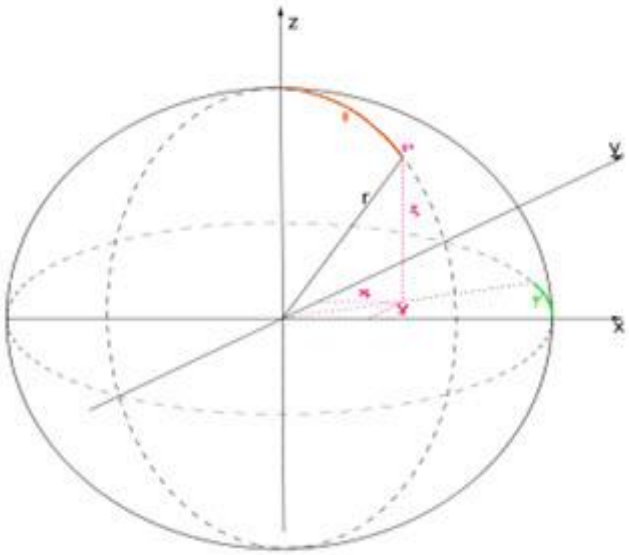
Geografske podatke lahko opredelimo kot podatke o opisnih, časovnih in kartografskih lastnostih ter odnosih med geografskimi objekti, katerih položaj (v literaturi s tega področja pogosto srečamo tudi izraz lokacija, ki ga bomo večkrat uporabili tudi mi) je podan v enotnem koordinatnem sistemu.
Geografski podatek je podatek, ki opisuje geografske pojave v prostoru. Vsebuje relacije med geografskimi objekti, s pomočjo katerih je mogoče pojav postaviti na določeno mesto v prostoru. Sestavljen je iz vsaj dveh delov, prostorske komponente, ki nam pove, kje se objekt nahaja in opisne komponente, ki nam pove, kaj in kakšen je objekt. Prostorska komponenta je lahko neposredna (npr. koordinate x, y, z) ali posredna (npr. hišna ali parcelna številka). Med opisnimi podatki so pomembne geometrične značilnosti geografskih objektov, kot so na primer oblika, velikost, dolžina in površina. Tipični geografski podatki so denimo zemljiški kataster, digitalni model reliefa, satelitski posnetki in digitalizirani topografski načrti, register prostorskih enot ipd.

Prostorsko komponento najpogosteje navedemo v georeferenčnem sistemu. Georeferenčni sistem navadno pojmujemo kot ustrezno določen koordinatni sistem.



Slika 2 Kartezični koordinatni sistem (http://www.fgg.uni-lj.si/~/sdrobne/GIS_Pojm/Index.htm)

Temeljna enota geografskih podatkov v dvodimenzionalnem primeru je koordinatni par (na Slika 2 Kartezični koordinatni sistem je to npr. par (58,32)), v prostorskem pa trojica (x, y, z). Položaj točke lahko opišemo v kartezičnem, geografskem koordinatnem sistemu ali v poljubnem koordinatnem sistemu. Geografski koordinatni sistem je sferni koordinatni sistem, kjer je referenčna ploskev rotacijski elipsoid, s pomočjo katerega določimo lego točk na krogli. Poglejmo si, kakšna je zveza med obema koordinatnima sistemoma. V geografskem koordinatnem sistemu položaj točk navajamo v sferičnih koordinatah. Ker torej geografski koordinatni sistem vsebuje sferične koordinate, lahko le-te pretvorimo v kartezični koordinatni sistem.



Slika 3 Geografski koordinatni sistem (http://sl.wikipedia.org/wiki/Sferni_koordinatni_sistem)

Geografske koordinate v koordinatnem sistemu dobimo s pomočjo normale na elipsoid v dani točki. Položaj točke določata geografski širina in dolžina na elipsoidu. Širino (ang. *Latitude*) predstavlja kot med normalo v dani točki in ravnino ekvatorja, dolžino (ang. *Longitude*) pa predstavlja kot med ravnino izhodiščnega meridijana in ravnino meridijana skozi dano točko. Vrednosti za geografsko širino in dolžino so izražene v ločnih stopinjah. V geografskem koordinatnem sistemu so za točko P, ki ima krajevni vektor, koordinate (glej Slika 3):

- razdalja točke P od izhodišča (oznaka r), tudi dolžina krajevnega vektorja $\vec{r}$ točke P;
- polarni kot (oznaka θ ali ϑ) je kot med pozitivno z-osjo in vektorjem $\vec{r}$;
- azimutni kot (oznaka φ) je kot med x-osjo in smerjo projekcije vektorja $\vec{r}$ na $(x - y)$ ravnino.

Tako zapišemo koordinate točke P na naslednji način (r, θ, φ). Kjer lahko

- krajevni vektor r zavzame poljubno veliko vrednost;
- polarni kot θ zavzame vrednosti med 0 in π (med 0° in 180°);
- azimut φ pa zavzame vrednosti met 0 in 2π (med 0° in 360°), merjeno v nasprotni smeri urinega kazalca.

Pretvorba iz sfernih koordinat v kartezične

$x = r\cos\theta \sin\rho \qquad y = r\sin\theta \sin\rho \qquad z = \cos\theta$

Pretvorba iz kartezičnih koordinat v sferne

$$r = \sqrt{x^2 + y^2 + z^2}$$

$$\rho = \begin{cases} \arccos\dfrac{x}{\sqrt{x^2+y^2}}, & za\ y \geq 0, \\ 2\pi - \arccos\dfrac{x}{\sqrt{x^2+y^2}}, & za\ y \geq 0; \end{cases}$$

$$\theta = \operatorname{arccot}\frac{z}{\sqrt{x^2+y^2}} = \frac{\pi}{2} - \arctan\frac{z}{\sqrt{x^2+y}}$$

Drugi del podatkov so metapodatki. To so podatki o podatkih. Metapodatki uporabnikom podajajo pomembne informacije o sestavi, vsebini, vrednosti, kakovosti, organizaciji, dostopnosti in možni uporabi shranjenih podatkov. Metapodatki so torej tehnična in poslovna interpretacija podatkov, ki podrobno opisujejo sestavo, obseg in vsebino geografskih podatkov. Kot primer metapodatkov imamo geografske podatke, kot so na primer ceste. Metapodatki zajemamo podatke o geografski obliki podatkov, torej cest, ki je lahko vektorska ali rastrska oblika. Zajemajo tudi podatke o območju koordinatnega prostora, v katerem se nahajajo ceste, o obliki, velikosti in dostopnosti datoteke, v kateri so shranjeni geografski podatki ceste. Zajemajo tudi podatke o vsebini podatkov, na primer število podatkov, kakšnega tipa so geografski podatki, v kakšnem koordinatnem sistemu so …

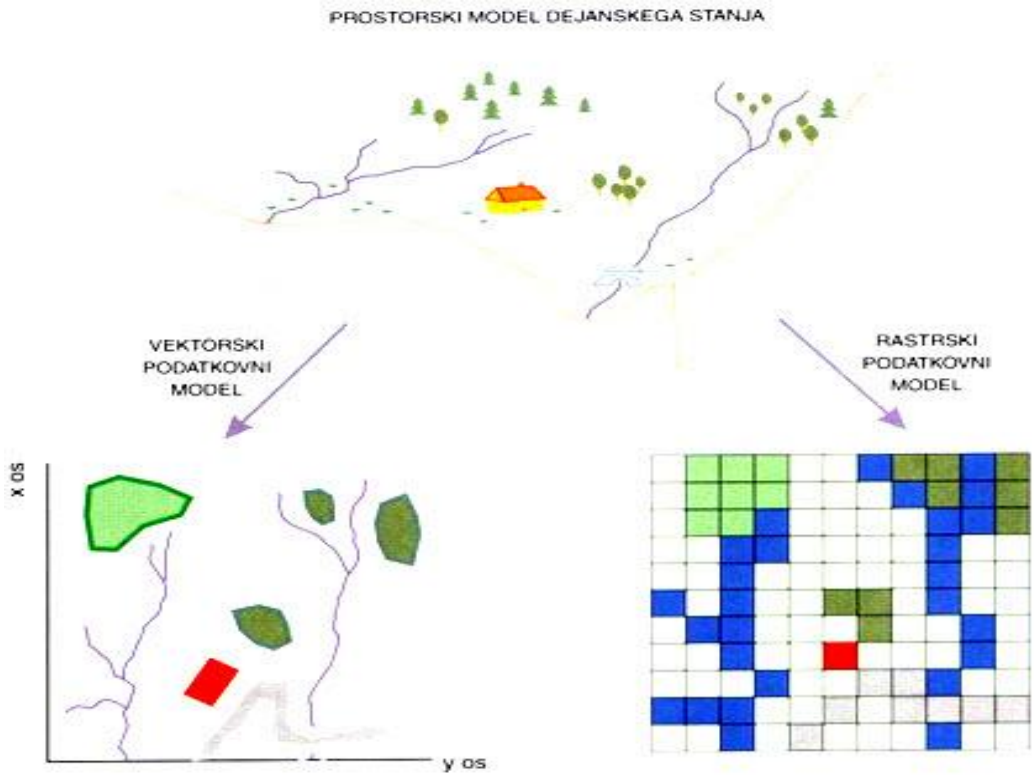
1.3 GEOGRAFSKI OBJEKTI

V splošnem lahko ločimo med fizičnimi in abstraktnimi objekti geografskega prostora. Fizični objekti obstajajo v fizični stvarnosti. Primeri so hiše, ceste, mostovi, utrdbe, gore, reke, morja itd. Abstraktni objekti so ustvarjeni v človeškem umu. Primeri abstraktnih objektov so denimo pravni objekti kot na primer zemljiška posest, politične in administrativne enote, razne planerske delitve prostora itd.

Sistem GIS simulira stvarno okolje z ustreznim modelom, ki ponazarja lokacijo, geometrijo, opisne značilnosti in razmerja med geografskimi pojavi. Tak model je na primer topografska karta. Na njej je lahko modeliran potek cest, z barvami označena višina terena itd. Geografski podatki so lahko grafično urejeni in predstavljeni na dva načina.

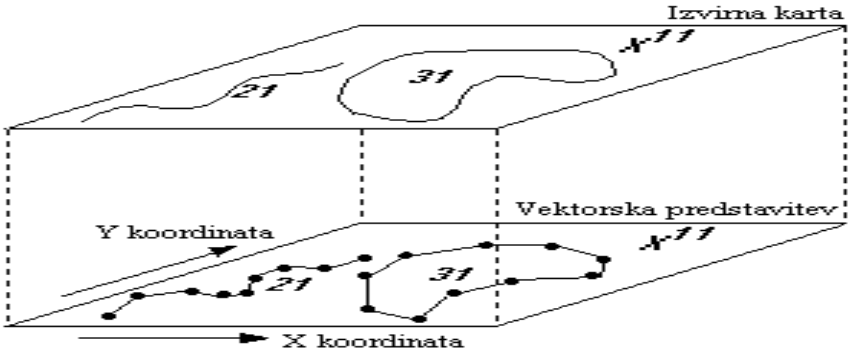
To sta vektorski in rastrski način:

- vektorski pristop temelji na modeliranju in upodobitvi geografskih pojavov v obliki točk, linij, in območij;
- rastrsko načelo temelji na modeliranju in upodobitvi geografskega prostora v obliki enakih in sistematično urejenih celic.



Slika 4 Predstavitev vektorskih in rastrskih podatkov (http://www.zrss.si/ppt/GEO_multplik_b.ppt)

Ker bomo v nadaljevanju uporabljali le vektorsko predstavitev, si bomo ogledali le-to in se z rastrsko predstavitvijo ne bomo ukvarjali. Vektorsko predstavitev objektov pogosto uporabljamo kot metodo predstavitve objektov na topografski karti. Slika 5 prikazuje, kako iz objektov na izvirni karti dobimo vektorsko predstavitev teh objektov. Tako je npr. jezero z izvirne karte vektorsko predstavljeno kot poligon.

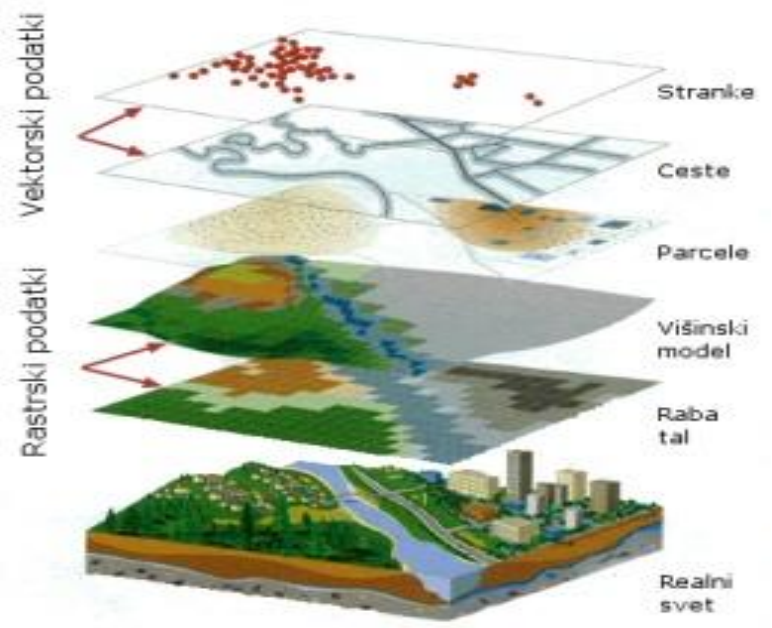


Slika 5 Predstavitev, kako iz objektov na izvirni karti ustvarimo vektorsko predstavitev geografskih objektov (http://www.fgg.uni-lj.si/~/sdrobne/GIS_Pojm/Index.htm)

1.4 KARTOGRAFSKI PODATKOVNI MODEL

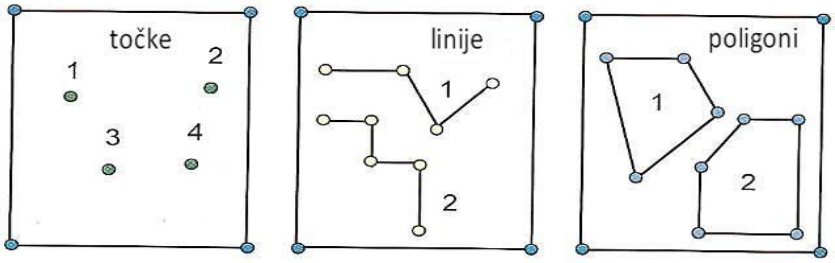
Podatke o več geografskih objektih lahko smiselno združimo v večjo logično enoto, ki jo imenujemo podatkovni sloj. Ta je lahko fizično izveden kot podatkovni niz ali kot datoteka oziroma kot skupina internih zapisov v bazi podatkov.

Podatkovni niz prikazuje lego naravnih in zgrajenih objektov v prostoru in nekatere njihove lastnosti. Vsebina podatkovnega niza je enaka vsebini topografske karte. Podatkovni model, ki je v rabi v sistemih GIS, temelji na razstavitvi vsebine topografske karte na ustrezne, po vsebini ločene, tematske plasti. Taki podatkovni sloji ločeno opisujejo relief, hidrologijo, izgrajene objekte, vegetacijo itd. kot sklop enakovrednih geografskih pojavov.



Slika 6 Prikaz kartografskega podatkovnega modela kot vsebinske podatkovne plasti (http://www.geoservis.si/uporabno/gis/gis.htm)

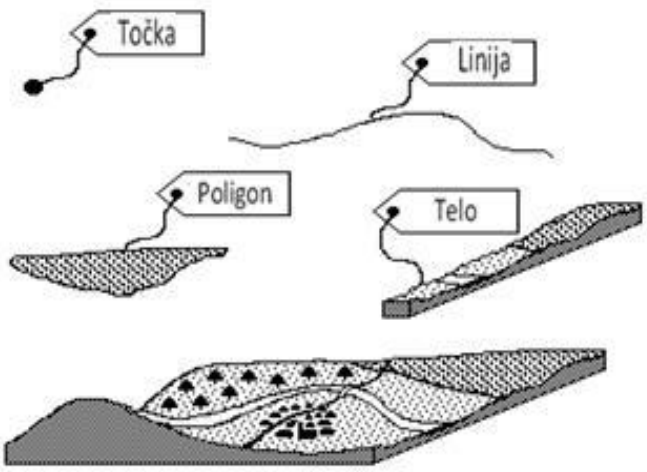
Na Slika 6 je prikazan kartografski podatkovni model kot vsebinske podatkovne plasti, ki se upodobijo v ravnini (2D). Sestavo modela stvarnega sveta navpično razstavimo oziroma, bolje rečeno, razločimo na tematske plasti, ki predstavljajo ustrezen podatkovni sloj. Podatkovni sloj predstavlja niz geografskih atributnih in grafičnih podatkov, ki opisujejo geografske značilnosti na obravnavanem geografskem območju. V sklopu takih tematskih plasti se lahko, glede na vsebovane geometrijske objekte, podatki razdelijo na točkovne (0D), linijske (1D) in območne (2D) vsebinske sloje, kar prikazuje Slika 7 Prikaz geometrijskih objektov.



Slika 7 Prikaz geometrijskih objektov(Tehnologija GIS, Šumrada, 2005, str. 111)

Grafični prikaz geografskega objekta lahko zgradimo s štirimi enostavnimi geometrijskimi objekti: točko, linijo, območjem in telesom. Povezava med dvema točkama predstavlja linijo. Več zaključenih povezav predstavlja območje (poligon), območja pa lahko sestavimo v telo. Nizi koordinatnih parov lahko predstavljajo neko linijo ali ploskev ter podajajo njun položaj in obliko. V GIS-u te enostavne geometrijske objekte uporabljamo za prikazovanje enodimenzionalnih, dvodimenzionalnih ali tridimenzionalnih geografskih objektov.

Točka je v GIS-u objekt brez dimenzije, ker nima dolžine in širine. Linija v GIS-u je enodimenzionalen geometrijski objekt, ker ima dolžino. Ploskev (poligon) je dvodimenzionalen geometrijski objekt in ima obseg in ploščino. Telo je v GIS-u tridimenzionalen geometrijski objekt, saj ima vključene tudi podatke o višini. Četrta dimenzija je v GIS-u navadno opredeljena kot čas. Slika 8 nam prikazuje grafični prikaz geografskega objekta, ki je zgrajen z **geometrijskimi objekti**. Vsak posamezen gradnik je v naravi predstavljen kot objekt. Kot že vemo, si geografske objekte lahko predstavljamo kot objekte v naravi. Torej objekt točka lahko v naravi predstavlja hišo, linija reko, ploskev (poligon) – npr. travnik, objekt telo pa predstavlja območje več geografskih objektov skupaj.



Slika 8 Prikaz geografskih objektov (http://www.fgg.uni-lj.si/~/sdrobne/GIS_Pojm/Index.htm)

1.5 OSNOVNI GEOGRAFSKI TIPI ISKANJA PODATKOV

Iskanje podatkov je v GIS-u zelo pogosta operacija, saj nas na karti vedno zanimajo podatki v obliki hišnega naslova, ulice, kraja, koordinat nekega objekta itd.

V GIS-u ločimo pet osnovnih tipov iskanja podatkov (glej Slika 9): v obliki kroga, pravokotnika, poligona ter iskanje s pomočjo linije in s pomočjo točke. Pri teh iskanjih bi radi preverili lego določene točke:

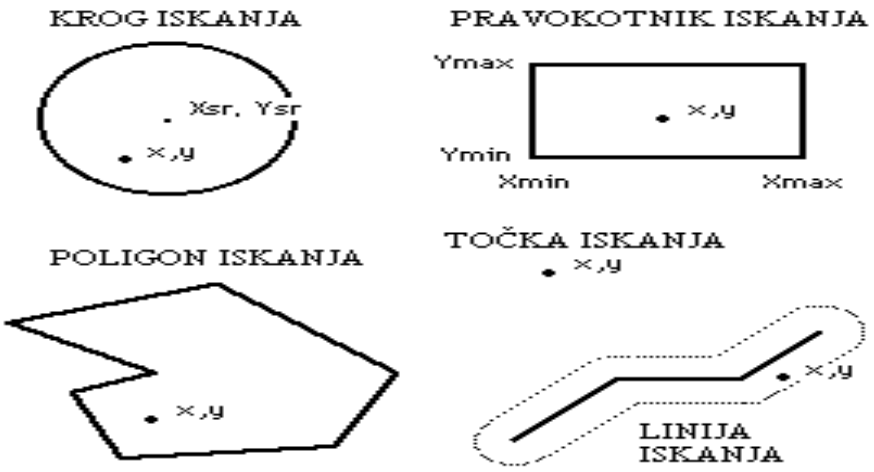
- poizvedovanje, ali točka s koordinatama x, y leži v krogu z radijem r. Preverimo, če velja enačba $(x-X_{sr})^2 + (y-Y_{sr})^2 \leq r^2$, kjer sta X_{sr} in Y_{sr} koordinati središča kroga;

- poizvedovanje, ali točka x, y leži znotraj pravokotnika. To preverimo tako, da ugotovimo, če velja $X_{min} \leq x \leq X_{max}$ ter $Y_{min} \leq y \leq Y_{max}$, kjer sta X_{min} in Y_{min} koordinati spodnjega levega oglišča, X_{max}, Y_{max} pa koordinati zgornjega desnega oglišča pravokotnika;

- poizvedovanje, ali točka leži znotraj poligona, izvedemo tako, da si zunaj poligona izberemo poljubno točko ter jo povežemo s točko v poligonu. S tem dobimo daljico, ki seka poligon. Sedaj prištejemo sečišča, in če je število sečišč neparno, leži točka v poligonu;

- poizvedovanje, ali je točka s koordinatama x in y dovolj blizu linije. Linija nam v realnosti lahko predstavlja cesto, reko ali most. Ti geografski objekti so v GIS-u navadno prestavljeni kot črta, ki določa središče teh geografskih objektov in vsebuje atribute širino, naklon itd. Z atributom širina določimo območje, ki je na Slika 9 predstavljeno s črtkano črto. S tem poizvedovanjem torej ugotavljamo, ali točka leži na neki cesti, reki itd. Ali je točka s koordinatami x in y dovolj blizu linije, preverimo tako, da točko povežemo pravokotno na linijo. Dolžina pravokotnice mora biti manjša od polovice vrednosti atributa širine. Čim krajša je ta pravokotnica, bliže liniji je iskana točka;

- poizvedovanje, ali se koordinate točke ujemajo s koordinatami neke druge točke, ki predstavlja nek objekt v geografskem informacijskem sistemu. Točka, ki predstavlja objekt, nam v realnosti predstavlja hišo, grad itd. Ti geografski objekti so v GIS-u navadno predstavljeni kot točka, ki določa središče teh geografskih objektov in vsebuje atribute širino, višino itd. Običajno ta tip poizvedovanja uporabljamo, ko želimo vedeti, ali miškin kurzor na zaslonu kaže na nek geografski objekt – točko. Pri primerjanju koordinat kurzorja s koordinatami v geografskem informacijskem sistemu upoštevamo odstopanja, ki so določena s širino geografskega objekta.

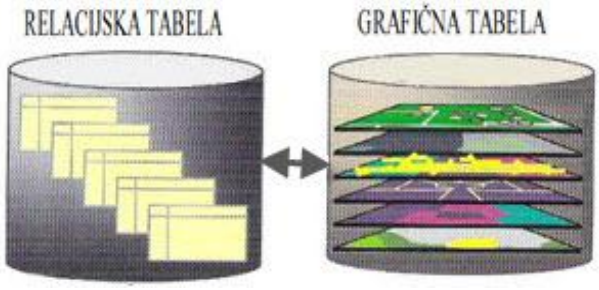


Slika 9 Geografski podatki za geografska poizvedovanja (http://www.fgg.uni-lj.si/~/sdrobne/GIS_Pojm/Index.htm)

1.6 SHRANJEVANJE GEOGRAFSKIH PODATKOV

V sistemu GIS so navadno opisni in geografski podatki o posameznem geografskem objektu izvedbeno in fizično shranjeni ločeno v dveh različnih tabelah, ki sta shranjeni v relacijski bazi podatkov. To sta grafična tabela z geografskimi podatki in relacijska tabela z opisnimi podatki. Da pa opisne in geografske podatke lahko med sabo povežemo, uporabljamo enolične identifikatorje geografskih pojavov. Ti služijo za medsebojno ločevanje posameznih geografskih objektov in hkrati omogočajo učinkovito povezavo opisnih in geografskih podatkov.

Kot smo povedali, podatki o geografskem objektu vsebujejo geografske in opisne podatke. Pri tem geografski podatki določajo lokacijo, obliko ter odnose med grafičnimi gradniki (točka, linija in poligon) in so shranjeni v grafični tabeli. V relacijski tabeli podatkov pa so shranjeni podatki, ki opisujejo lastnosti geometrijskih objektov. Geografski in opisni podatki so neposredno povezani preko enolične določene številke elementa v prostoru (identifikator).

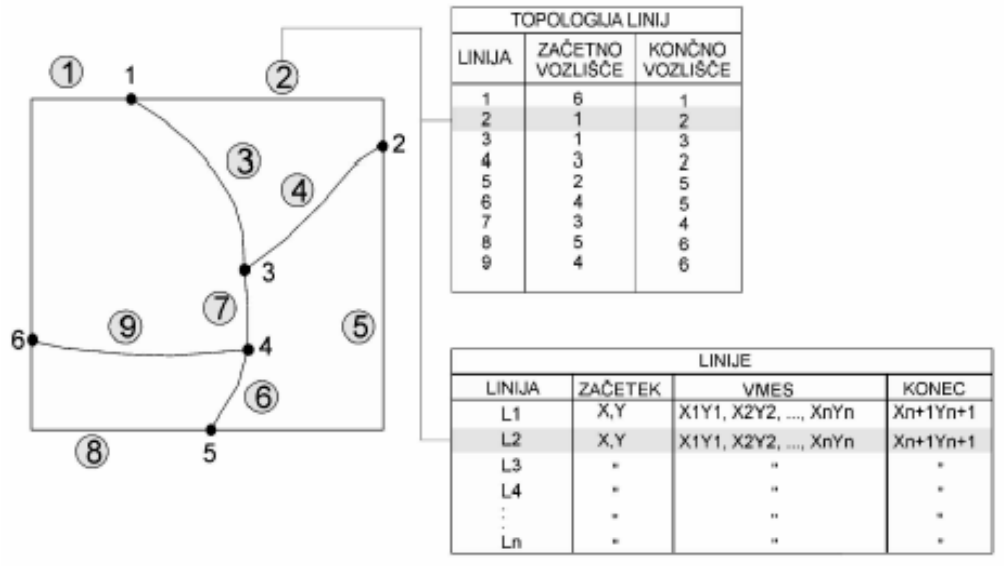


Slika 10 Prikaz shranjevanja opisnih in geografskih podatkov (Tehnologija GIS, Šumrada, 2005, str. 112)

Obnovimo na kratko osnovne značilnosti relacijskih baz podatkov. Osnovna enota v relacijski bazi podatkov je tabela, v kateri vrstica opredeljuje geografski objekt, stolpec pa atribut. Večina relacijskih baz podatkov je sestavljena iz več tabel. Relacijska struktura podatkov omogoča različne povezave med tabelami. Te izvajamo s pomočjo primarnega in tujega ključa relacijske tabele. S temi povezavami informacije ene tabele povežemo z informacijami v drugi. Tako se izognemo podvajanju istih vnosov podatkov.

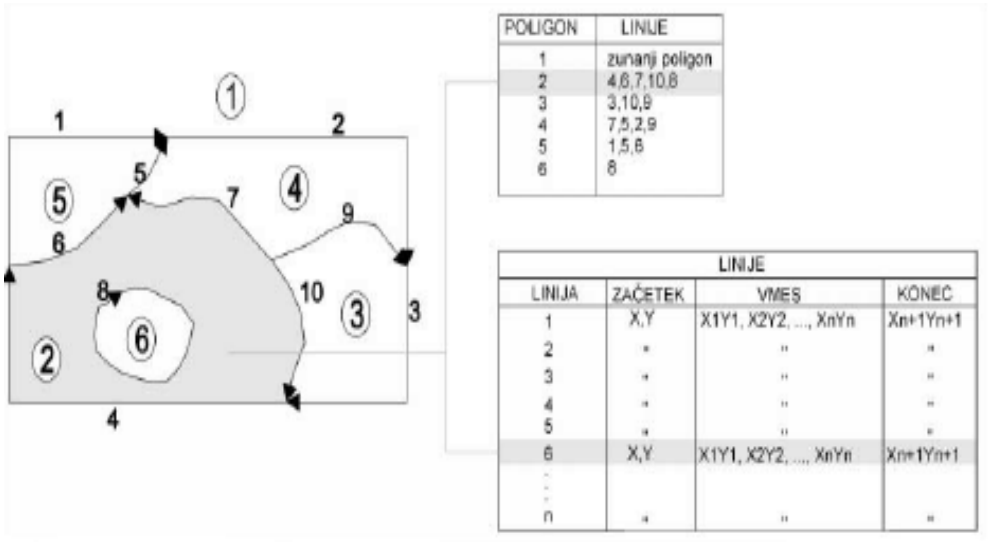
Prednost grafične tabele je v grafičnem vnašanju podatkov, ki omogoča vnos geografskih pozicij. Ti podatki vsebujejo tudi določene podatkovne tipe in so shranjeni v grafični tabeli na način, ki omogoča optimalno obdelavo geografskih podatkov. Slabost uporabe grafične tabele pa je v slabi povezanosti obeh vrst podatkov. Tako je treba pri pregledovanju podatkov nekega objekta poizvedovati po obeh tabelah podatkov. Čas poizvedovanja v grafični tabeli lahko skrajšamo z omejitvijo količine obravnavanih geografskih podatkov.

Poglejmo si primer shranjevanja geografskih podatkov v grafični tabeli in opisnih podatkov v relacijski tabeli. Tabela topologija linij je relacijska (z opisnimi podatki), tabela linije pa grafična (z geografskimi podatki). Na Slika 11 vidimo primer geografskega prostora, ki vsebuje določene geografske objekte. Te so ceste, ki so v GIS-u navadno predstavljene kot linije. V GIS bi radi shranili podatke o cestah. Geografske podatke, kot so položaj začetne, končne točke ter položaj vmesnih točk hranimo v grafični tabeli linije. Tam so shranjeni v obliki koordinatnih parov (x, y). Druge lastnosti teh objektov (npr. številka začetne in končne točke) pa shranimo v relacijsko tabelo topologija linij. S pomočjo ključev oz. identifikatorjev zagotovimo povezavo med ustreznimi tabelami v relacijski bazi.



Slika 11 Primer zapisa grafičnega gradnika (linija) v dve ločeni bazi podatkov (http://www.geo-zs.si/publikacije_arhiv/Clanki/Geologija_45_2/sinigoj_etal_45_2.pdf)

Poglejmo si še en primer. Na Slika 12 vidimo primer geografskega prostora, ki vsebuje geografske objekte – v našem primeru meje parcel. Meje, ki omejujejo parcele, so navadno v GIS-u predstavljene kot zaključene linije, ki predstavljajo poligon. V GIS bi radi shranili podatke o mejah parcel. Geografske podatke, kot so položaj začetne, končne točke ter položaj vmesnih točk parcel hranimo v grafični tabeli linije. Tam so shranjeni v obliki koordinatnih parov (x, y). Druge lastnosti teh objektov (npr. številke točk, ki predstavljajo poligon) pa shranimo v relacijsko tabelo. S pomočjo ključev oz. identifikatorjev zagotovimo povezavo med ustreznimi tabelami v relacijski bazi.



Slika 12 Primer zapisa grafičnega gradnika (poligon) v dve ločeni bazi podatkov (http://www.geo-zs.si/publikacije_arhiv/Clanki/Geologija_45_2/sinigoj_etal_45_2.pdf)

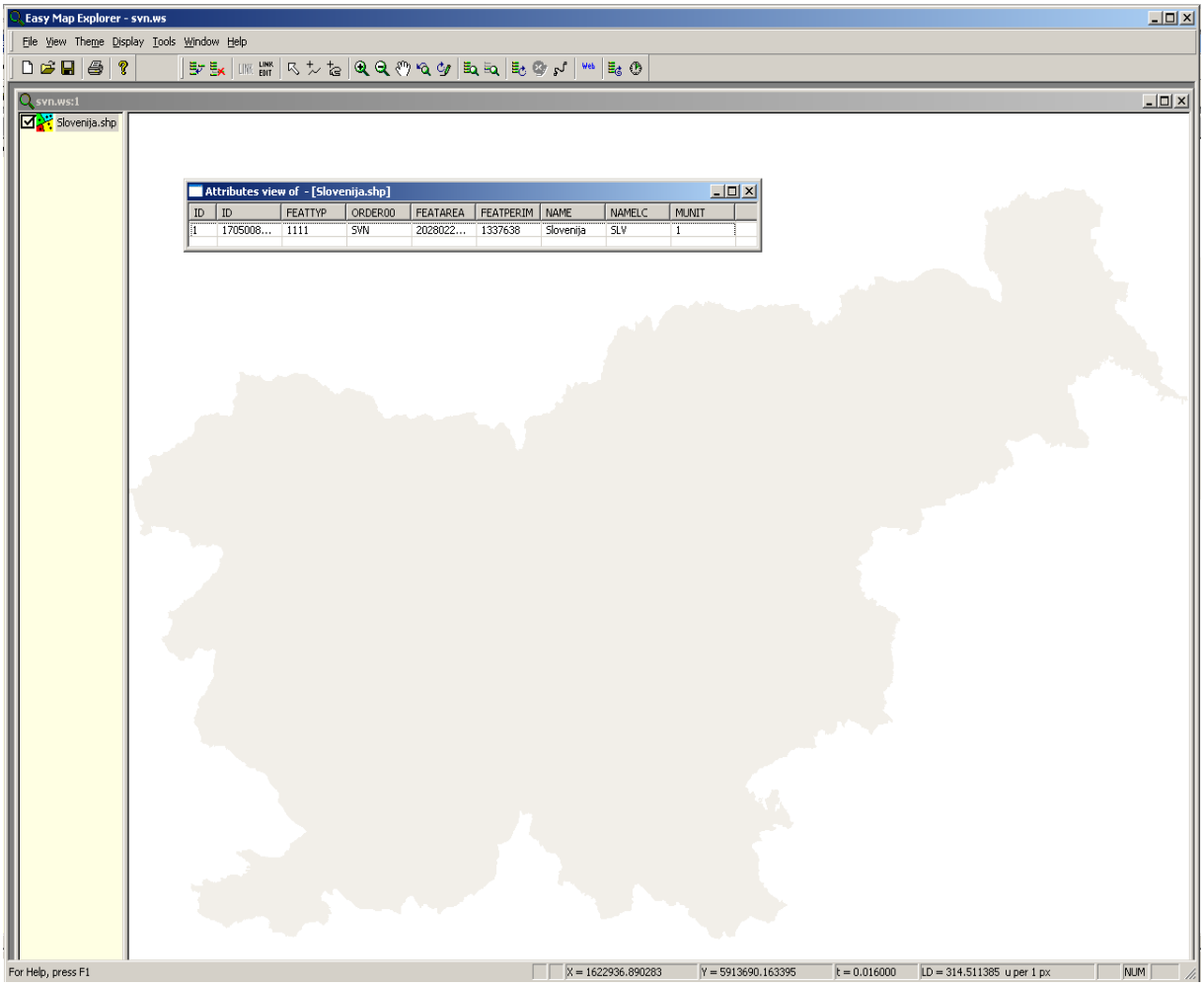
1.7 GIS ORODJA ZA OBDELAVO GEOGRAFSKIH PODATKOV

Do sedaj smo govorili o splošnih pojmih geografskega informacijskega sistema. Vendar smo v definiciji omenili, da tehnologijo GIS tvorijo tudi orodja, ki jih potrebujemo za delo z geografskimi podatki. Za grafični prikaz geografskih objektov uporabimo orodja GIS, kot so na primer ArcGIS, ArcMap, ArcCaralog, MapInfo, Easy Map Explorer in podobni. V GIS orodju za prikaz podatkovnega modela potrebujemo tri vrste datotek.

Te so:

- **SHP** (ang. *shape*) predstavlja obliko geografskega objekta;
- **DBF** (ang. *data base file*) vsebuje bazo podatkov;
- **SHX** (ang. *shape index*) indeksna datoteka za povezovanje SHP in DBF datotek.

Poglejmo si primer uporabe GIS orodja **Easy Map Explorer**. Radi bi pokazali geografski objekt Slovenija. Datoteka SHP bo poskrbela za grafično predstavitev (tam je Slovenija predstavljena kot poligon), lastnosti Slovenije kot geografskega objekta (št. prebivalcev, površina, ime države itd.) pa so shranjeni v datoteki DBF (na Slika 13 prikazani v oknu z naslovom Atributes view of – [Slovenija.shp]). Za povezovanje obeh navedenih datotek se uporablja datoteka SHX z enakim imenom, kot sta bili navedeni datoteki.



Slika 13 Prikaz datotek Slovenja.shp, Slovenija.dbf in Slovenija.shx v programu Easy Map Explorer

2 POSTGRESQL Z DODATKOM POSTGIS

V tem poglavju si bomo ogledali določene posebnosti, ki jih ima sistem za upravljanje z relacijskimi bazami podatkov v PostgreSQL-u v povezavi z GIS. Nekatere od teh posebnosti bodo prikazane na primerih.

PostgreSQL je prosto dostopen sistem za upravljanje z objektno – relacijsko podatkovno bazo (ORDBMS). Razvili so ga na oddelku za računalniške znanosti Berkeley Univerze v Kaliforniji. Pri razvoju PostgreSQL so kot prvi uporabili mnoge koncepte, ki so kasneje postali sestavina v številnih sistemih za upravljanje podatkovnih baz. PostgreSQL podpira velik del standarda jezika SQL in ponuja veliko značilnosti, ki odlikujejo sodobne sisteme za uporabljanje baz podatkov. PostgreSQL sam po sebi ne omogoča hranjenje geografskih podatkov kot objektov v bazi. Zato si bomo v nadaljevanju ogledali razširitev sistema PostgreSQL, imenovano PostGIS, ki omogoča hranjenje geografskih podatkov.

PostGIS je dodatek k sistemu za delo z bazami podatkov PostgreSQL, ki omogoča, da v bazi hranimo geografske podatke. PostGIS podpira tudi vse funkcije za analizo in obdelavo geografskih podatkov, ki so definirane s strani organizacije OpenGIS Consortium (OGC). Z dodatkom PostGIS pridobimo naslednje podatkovne tipe (tabela), ki jih sam PostgreSQL ne pozna.

ŠT.	GEOMETRIJSKI TIP
0	geometry
1	point
2	linestring
3	polygon
4	multipoint
5	multilinestring
6	multipolygon

To nam omogoča, da v določenem stolpcu (za nas bo imenovan z imenom `the_geom`) lahko hranimo na primer podatek tipa točka (`Point`). Stolpec `the_geom` je tako imenovani geometrijski stolpec in je tipa `geometry`, ki je shranjen v vsaki grafični tabeli. Stolpec `the_geom` vsebuje vse podatke o geometriji geografskih podatkov, na primer vsebuje podatke o geometriji neke točke. Več o njem bomo izvedeli v nadaljevanju.

OpenGIS Consortium (OGC) je mednarodna neprofitna organizacija. Njen namen je razvoj industrijskih standardov, ki omogočajo povezljivost različnih sistemov GIS. Glavni cilj te organizacije je razvoj odprte specifikacije GIS (OGIS – *OpenGIS Interface Specification*). Ta omogoča izgradnjo vmesnika, preko katerega je mogoče doseči učinkovito povezavo med različnimi programskimi orodji v različnih GIS. Tehnologija OpenGIS je zasnovana tako, da zagotavlja enostaven dostop do raznovrstnih geografskih podatkov po omrežju.
Pod imenom OpenGIS (projekt OpenGIS) se torej skrivajo trije osnovni sestavni deli:

- OpenGIS kot skupek tehnologij, ki omogoča obdelavo geografskih podatkov;
- OpenGIS specifikacija (OGIS) omogoča povezavo med sistemi za delo z geografskimi podatki;
- Open GIS Consortium je razvoju odprtih GIS tehnologij in integraciji obdelav geografskih podatkov posvečen konzorcij.

V prejšnjem razdelku smo povedali, da so opisni in geografski podatki v sistemu GIS shranjeni v dveh ločenih podatkovnih tabelah v isti relacijski bazi. To sta grafična in relacijska tabela podatkov. Grafična tabela omogoča vnos geografskih pozicij, ki se shranijo v geometrijskem stolpcu `the_geom` tipa `geometry`. V relacijski tabeli so shranjeni podatki, ki opisujejo lastnosti geometrijskih objektov. Da je upravljanje z geografskimi in opisnimi podatki čim bolj enostavno in pregledno, poskrbi sistem

za upravljanje podatkovnih baz (SUPB). V primerih, opisanih v nadaljevanju, je ta sistem PostgreSQL z dodatkom PostGIS.

PostGIS za SQL definira standardne tipe geografskih objektov in funkcije, ki so potrebne za delo s temi objekti, o katerih smo že govorili. PostGIS definira tudi dve tabeli, ki vsebujeta meta podatke in se ustvarita v vsaki zgrajeni relacijski bazi podatkov. Ti tabeli sta poimenovani `spatial_ref_sys` in `geometry_columns`. Tabela `spatial_ref_sys` vsebuje podatke o koordinatnih sistemih in nam omogoča koordinatno transformacijo geografskih podatkov. Če tabela ne vsebuje podatkov o željenem koordinatnem sistemu, lahko ustrezne podatke o njem dodamo v tabelo. Tabela `geometry_columns` vsebuje podatke le o tistih tabelah, ki so vsebovane v isti relacijski bazi, kot je tabela `geometry_columns`. Poglejmo si zgradbo in uporabo teh tabel.

Tabela spatial_ref_sys vsebuje numerične podatke, s katerimi se sklicujemo na ustrezne koordinatne sisteme (njihove oznake `srid`) in tekstovne opise koordinatnih sistemov, ki so uporabljeni v grafični tabeli podatkov. Povejmo še enkrat, da se tabela avtomatično ustvari in napolni s podatki o koordinatnih sistemih takrat, ko ustvarimo novo bazo. Če tabela ne vsebuje podatkov o željenem koordinatnem sistemu, podatke o novem koordinatnem sistemu lahko dodamo z ustreznim ukazom.

Poglejmo si zgradbo in nekaj vsebine tabele. Tabela `spatial_ref_sys` je zgrajena na naslednji način:

```
CREATE TABLE spatial_ref_sys (srid INTEGER NOT NULL PRIMARY KEY, auth_name
VARCHAR(256), auth_srid INTEGER, srtext VARCHAR(2048), proj4text
VARCHAR(2048));
```

Sedaj si poglejmo nekaj vsebine podatkov, ki so v tabeli `spatial_ref_sys`. V stolpcu `srid` so identifikacijske številke koordinatnih projekcij. V ostalih tabelah se bomo sklicevali na te številke in s tem povedali, v katerem koordinatnem sistemu so zapisani podatki o določenem geometrijskem objektu. V stolpcih `proj4text` in `srtext` so shranjeni vsi podatki o koordinatnih sistemih.

srid	auth_name	auth_srid	proj4text	srtext
4002	EPSG	4002	+proj=longlat +a=6377340.189 +b=6356034.447938534 +no_defs	GEOGCS["Unknown datum based upon the Airy Modified 1849 ellipsoid",DATUM["Not_specified_based_on_Airy_Modified_1849_ellipsoid" …
4005	EPSG	4005	+proj=longlat +a=6377492.018 +b=6356173.508712696 +no_defs	GEOGCS["Unknown datum based upon the Bessel Modified ellipsoid",DATUM["Not_specified_based_on_Bessel_Modified_ellipsoid",…
4007	EPSG	4007	+proj=longlat +a=6378293.645208759 +b=6356617.987679838 +no_defs	GEOGCS["Unknown datum based upon the Clarke 1858 ellipsoid",DATUM["Not_specified_based_on_Clarke_1858_ellipsoid",…
4009	EPSG	4009	+proj=longlat +a=6378450.047548896 +b=6356826.621488444 +no_defs	GEOGCS["Unknown datum based upon the Clarke 1866 Michigan ellipsoid",DATUM["Not_specified_based_on_Clarke_1866_Michigan_ellipsoid",…
4010	EPSG	4010	+proj=longlat +a=6378300.789 +b=6356566.435	GEOGCS["Unknown datum based upon the Clarke 1880 (Benoit) ellipsoid",DATUM["Not_specified_base

			+no_defs	d_on_Clarke_1880_Benoit_ellipsoid",…
4326	EPSG	4326	+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs	'GEOGCS["WGS84",DATUM["WGS_1984",SPHEROID["WGS84",6378137,298.257223563,…

Pri našem delu bomo pogosto potrebovali koordinatno transformacijo geografskih podatkov med Lambertovo projekcijo, imenovano tudi WGS84[1], in transverzalno Mercatorjevo projekcijo.[2]

Geografski podatki, ki jih želimo obdelati, so večinoma shranjeni v Lambertovi projekciji, ki jih transformiramo v transverzalno Mercatorjevo projekcijo. Ker tabela `spatial_ref_sys` privzeto vsebuje le podatke o Lambertovi projekciji, ne pa tudi o transverzalni Mercatorjevi projekciji, je treba te podatke še dodati. Poglejmo si podatke o obeh projekcijah:

- Lambertova projekcija (WGS84) je enoten geografski koordinatni sistem, ki vsebuje zemljepisno širino in zemljepisno dolžino. Koordinate podatkov so podane v decimalnih stopinjah. Poglejmo si, kako so bili dodani podatki o Lambertovi projekciji v tabelo `spatial_ref_sys`.

```
INSERT INTO spatial_ref_sys (srid, auth_name, auth_srid, proj4text,
srtext) VALUES (4326, 'spatialreference.org', 4326, '+proj=longlat
+ellps=WGS84 +datum=WGS84 +no_defs ',
'GEOGCS["WGS84",DATUM["WGS_1984",SPHEROID["WGS84",6378137,298.2572235
63,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenw
ich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.01745329251994328,AU
THORITY["EPSG","9122"]],AUTHORITY["EPSG","4326"]]');
```

Ta projekcija ima torej identifikacijsko številko `srid` z vrednostjo `4326`. To vrednost bomo v nadaljevanju uporabljali za identifikacijo tega koordinatnega sistema.

- Transverzalna Mercatorjeva projekcija pa ima koordinate podatkov podane v metrih. Ker podatki o transverzalni Mercatorjevi projekciji še niso vsebovani v tabeli `spatial_ref_sys`, je te treba dodati. To storimo z ukazom

```
INSERT INTO spatial_ref_sys VALUES (54004, 'EPGS', 54004, '', '+proj
= merc +lat_ts=0 +lon_0=0 +k=1.000000 +x_0=0 +y_0=0 +ellps=WGS84 +
datum=WGS84 +units=m no_defs');
```

Sedaj imamo v `tabeli spatial_ref_sys` vsebovane tudi podatke o transverzalni Mercatorjevi projekciji, katere bomo potrebovali za tranformacijo koordinat. Na to projekcijo se bomo sklicevali s `srid` z vrednostjo `54004`.

[1] Lambertova projekcija ali WGS84 – je globalni koordinatni sistem z izhodiščem v težišču Zemlje in se vrti skupaj z njo. Položaj točke v tem sistemu je določen s kartezičnimi koordinatami (X, Y, Z) ali z geografskimi koordinatami (fi, lambda, h; geografska širina, dolžina in elipsoidna višina). Višina v tem sistemu je določena kot pravokotna oddaljenost točke nad WGS84 elipsoidom, ki aproksimira Zemljo kot telo.

[2] Transverzalna Mercatorjeva projekcija – je pokončna, konformna, cilindrična projekcija. Skratka, je pravokotni kartezični koordinatni sistem. Pri tem črte pravokotne koordinatne mreže tvorijo kvadrate in so na topografskih kartah narisane v medsebojni oddaljenosti celih kilometrov.

srid	auth_name	auth_srid	proj4text	srtext
54004	EPSG	54004	+proj=merc +lat_ts=0 +lon_0=0 +k=1.000000 +x_0=0 +y_0 = 0 +ellps = WGS84 + datum = WGS84 +units = m no_defs	

Tabela geometry_columns, ki jo tudi definira PostGIS, vsebuje podatke o tistih tabelah v bazi, ki vsebujejo geometrijske stolpce (torej geometrijske podatke) in je zgrajena na naslednji način:

```
CREATE TABLE geometry_columns (f_table_catalog VARRCHAR(256) NOT NULL,
f_table_schema VARCHAR(256) NOT NULL, f_table_nam VARCHAR(256) NOT NULL,
f_geometry_column VARCHAR(256) NOT NULL, coord_dimension INTEGER NOT NULL,
srid INTEGER NOT NULL, type VARCHAR(30) NOT NULL);
```

Vsebina podatkov, ki so v tabeli `geometry_column`:

- imena tabel vsebovanih v bazi;
- ali ima posamezna tabela geometrijski stolpec;
- kakšna je dimenzija koordinat geografskih podatkov v posameznih tabelah;
- podatke o vrsti koordinatnega sistema posameznih tabel;
- kašnega geometrijskega tipa so podatki v posamezni tabeli.

Poglejmo si primer. Denimo, da imamo v relacijski bazi tabeli **ceste** in **drzava**. Tabela **ceste** vsebuje podatke o geometriji cest, ki so shranjeni v stolpcu `the_geom`. Tabela **drzava** pa vsebuje podatke o geometriji države, ki so prav tako shranjeni v stolpcu z imenom `the_geom`. Ker je cesta v GIS-u predstavljena kot linija, je geometrijskega tipa `LineString`. Država je predstavljena kot poligon, zato je geometrijskega tipa `MultiPolygon`. Tabela **geometry_column**, ki je vsebovana v tej isti bazi kot dani tabeli, vsebuje podatke o teh dveh tabelah. Vidimo, da so v tabelah shranjeni geografski podatki, saj obe tabeli vsebujeta geometrijski stolpec, ki se obakrat imenuje `the_geom`. Vidimo tudi, da so podatki v danih tabelah v Lambertovi projekciji (imajo `srid` vrednost `4326`) in so geometrijskega tipa `LineString` ter `MultiPolygon`.

f_table_catalog	f_table_schema	f_table_name	f_geometry_column	coord_dimension	srid	type
	public	ceste	the_geom	2	4326	LINESTRING
	public	drzava	the_geom	2	4326	MULTIPOLYGON

PostGIS uporablja knjižnico `proj4`, ki vsebuje funkcije za koordinatne transformacije geografskih podatkov. Mi bomo uporabljali koordinatno transformacijo med Lambertovo projekcijo, imenovano tudi WGS84, v kateri so vhodni geografski podatki, in transverzalno Mercatorjevo projekcijo, v kateri bodo izhodni geografski podatki.

Način transformacije geografskih podatkov si oglejmo na primeru. Radi bi naredili koordinatno transformacijo geografskih podatkov med Lambertovo projekcijo in transverzalno Mercatorjevo projekcijo. Koordinatno transformacijo uporabimo v funkciji `vnosSHPvPG` (glej stran 48), ki uvozi datoteko z geografskimi podatki v relacijsko bazo. Dano imamo tabelo `tabela`, ki vsebuje geografske podatke. Podatki o geometriji teh podatkov so shranjeni v stolpcu `the_geom` geometrijskega tipa `geometry`. Dani podatki, ki jih želimo transformirali v transverzalno Mercatorjevo projekcijo, so shranjeni v Lambertovi projekciji. Transformacijo danih geografskih podatkov storimo tako, da izvedemo SQL ukaza:

```
ALTER TABLE " + tabela + " DROP CONSTRAINT enforce_srid_the_geom;
UPDATE " + tabela + " SET the_geom = transform(the_geom, "  + mercatorProj
+  ");"
```

Najprej odstranimo na tabeli `tabela` ključ `enforce_srid_the_geom`, ker nam na tabeli omejuje spreminjanje podatkov. Nato naredimo transformacijo koordinat iz Lambertove projekcije v transverzalno Mercatorjevo projekcijo. Za transformacijo koordinat uporabimo funkcijo **transform**, ki je definirana v knjižnici `proj4`. Funkcija potrebuje za vhodne podatke o geometriji geografskih podatkov, ki jih transformiramo – to so naši dani podatki, in `srid` podatek, s katerim se sklicujemo na ustrezen koordinatni sistem, v katerega transformiramo podatke. Ker želimo transformirati dane podatke v transverzalno Mercatorjevo projekcijo, uporabimo `srid` vrednost `54004`. Ta vrednost je shranjena v spremenljivki `mercatorProj`.

2.1 DIMENZIONALNO RAZŠIRJEN MODEL

Rekli smo, da PostGIS vsebuje tudi vse funkcije za obdelavo geografskih podatkov, kot jih definira OpenGIS. Eno med njimi, **transform**, ki omogoča preslikavo med koordinatnimi sistemi, smo spoznali že v prejšnjem razdelku. Med njimi so zelo pomembne tudi funkcije, ki definirajo prostorske odnose med objekti. Oglejmo si kar v splošnem, kako so s strani OpenGIS ti odnosi definirani.

V razdelku 1.4, ko smo govorili o kartografskem podatkovnem modelu, smo omenili, da se v sklopu tematskih plasti podatki razdelijo glede na vsebovane geometrijske objekte. V GIS-u se geometrijski objekti uporabljajo za prikazovanje enodimenzionalnih, dvodimenzionalnih ali tridimenzionalnih geografskih objektov. Da si malo osvežimo spomin, povejmo še enkrat, da je točka geometrijski objekt brez dimenzije, linija enodimenzionalen, ploskev dvodimenzionalen in telo tridimenzionalen geometrijski objekt. Prostorski odnosi, ki jih opisuje dimenzionalno razširjen model, so: *Disjoint* (disjunktno), *Touches* (dotikanje), *Crosses* (križanje), *Within* (leži znotraj), *Overlaps* (prekrivanje) in *Intersects* (sekanje).

Dimenzionalno razširjen model je matematični pristop, ki definira prostorske odnose med geometrijskimi objekti različnih tipov in dimenzij in temelji na definicijah notranjosti, meja in zunanjega dela geometrijskih objektov.

V tabeli so prikazane definicije splošne oblike dimenzionalnega razširjenega modela. Za dani geometrijski objekt A predstavlja I(A) notranjost, B(A) mejo ter E(A) zunanjost objekta A. Funkcija Dim(x) nam da vrednosti –1, 0, 1 ali 2, ki pomenijo dimenzijo geometrijskega objekta x. Pri tem velja:

- –1 označuje nedefiniran geometrijski objekt;
- 0 označuje objekt brez dimenzije (točka);
- 1 označuje enodimenzionalen geometrijski objekt (linija);
- 2 označuje dvodimenzionalen geometrijski objekt (poligon).

Presek delov (meje, notranjosti, zunanjosti) dveh geometrijskih objektov A in B je lahko niz geometrijskih objektov mešanih dimenzij. Na primer presek mej dveh poligonov je lahko nekaj točk in linij.

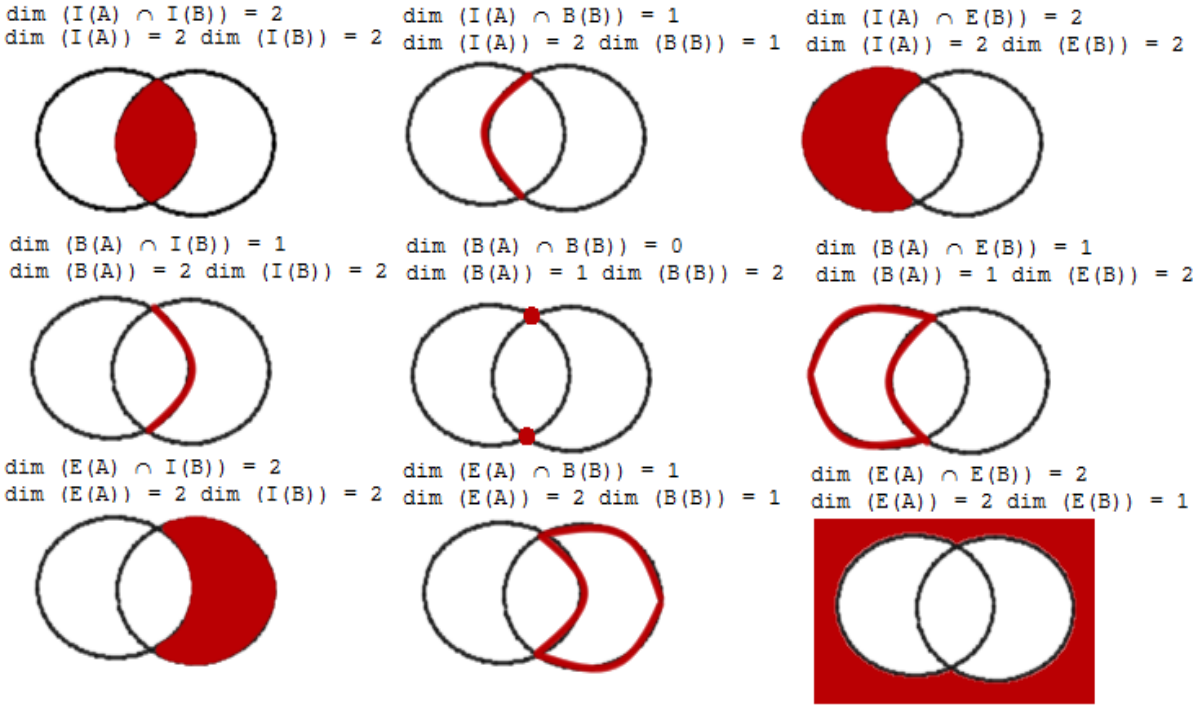
Denimo, da imamo dva geometrijska objekta. Glede na njuno lego se lahko:

- dotikata (*Touches*);
- križata (*Crosses*);
- prekrivata (*Overlaps*);
- sekata (*Intersect*);
- leži znotraj (*Within*);
- disjunktna (*Disjoint*).

Njun odnos lahko razberemo iz vrednosti, ki so v tabeli, prikazani spodaj.

	NOTRAJNOST (I)	**MEJA (B)**	**ZUNAJNOST (E)**
NOTRAJNOST (I)	dim(I(A)∩ I(B))	dim(I(A) ∩ B(B))	dim(I(A) ∩ E(B))
MEJA (B)	dim(B(A) ∩ I(B))	dim(B(A) ∩ B(B))	dim(B(A) ∩ E(B))
ZUNAJNOST (E)	dim(E(A) ∩ I(B))	dim(E(A) ∩ B(B))	dim(E(A) ∩ E(B))

Primere vrednosti dimenzij presekov iz tabele si poglejmo na primerih. Primeri so predstavljeni z geometrijskima objektoma A in B, ki sta krožnici in se sekata. Kot smo že omenili, je krožnica dvodimenzionalen geometrijski objekt in ima dimenzijo 2.



Slika 14 Primeri računanja dimenzij preseka dveh geometrijskih objektov

Navedimo sedaj točno definicijo omenjenih prostorskih odnosov. V teh definicijah bomo za primer brez dimenzionalne geometrije uporabljali geometrijske objekte *Point* (točka) in *MultiPoint* (zbirka točk), za enodimenzionalne geometrije *LineStrings* (linija) in *MultiLineStrings* (zbirka linij), za sklicevanje na dvodimenzionalne geometrije pa *Polygon* (poligon) in *MultiPolygons* (zbirka poligonov).

Za vsak prostorski odnos med dvema geometrijskima objektoma so vrednosti v zgoraj omenjeni tabeli različne. V tabeli bomo uporabljali naslednje oznake:

- F označuje nedefiniran geometrijski objekt, torej mora na tem mestu biti vrednost -1, kar pomeni, da dimenzije preseka ni;
- T označuje 0, 1 ali 2-dimenzionalne geometrijske objekte, torej so v tabeli na tem mestu vrednosti 0, 1 ali 2;
- * označuje –1, 0, 1 ali 2-dimenzionalne geometrijske objekte, torej so vrednosti dimenzije ustreznega preseka lahko poljubne.

2.1.1 DISJUNKTNOST GEOMETRIJSKIH OBJEKTOV

Za dana geometrijska objekta A in B preverimo, ali sta disjunktna (sta v odnosu *Disjoint*) tako, da preverimo, ali velja $A \cap B = \varnothing$. To pomeni, da mora notranjost obeh objektov imeti prazen presek.

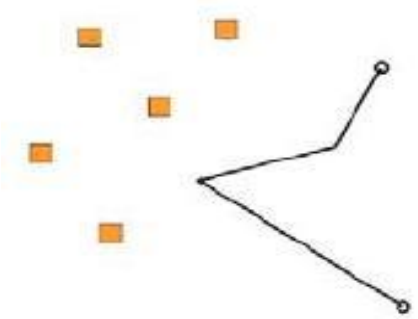
Torej mora veljati:

- objekta se ne smeta dotikati ali sekati;
- notranjost prvega geometrijskega objekta mora imeti prazen presek z robom drugega objekta;
- notranjost drugega geometrijskega objekta mora imeti prazen presek z robom prvega objekta.

Da sta dana geometrijska objekta v odnosu *Disjoint*, morajo za ta dva objekta vrednosti v tabeli torej ustrezati vzorcu:

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	F	F	*
Meja (A)	F	F	*
Zunanjost (A)	*	*	*

Poglejmo si primer na Slika 15, kjer je A geometrijski objekt linija in B *MultiPoint* (zbirka točk). Geometrijska objekta se ne dotikata in tudi ne sekata. Vemo, da je točka brezdimenzionalen geometrijski objekt in ima dimenzijo 0, linija pa enodimenzionalen geometrijski objekt in dimenzijo 1.



Slika 15 Odnos Disjoint (http://www.gis.hsr.ch/wiki/images/3/3d/9dem_springer.pdf)

Oglejmo si še, kako bi izpolnili celotno tabelo. Ker se objekta ne dotikata in tudi ne sekata, je dimenzija preseka notranjosti geometrijskih objektov −1 (−1 označuje dimenzijo nedefiniranega geometrijskega objekta). Dimenzija preseka notranjosti prvega geometrijskega objekta (linije) in meje drugega geometrijskega objekta (zbirke točk) je torej −1, ker je njun presek prazen. Presek meje prvega geometrijskega objekta (linije) in notranjosti drugega geometrijskega objekta (zbirke točk) je prav tako prazen, zato zapišemo v tabelo vrednost −1. Tudi dimenzija preseka meje prvega geometrijskega objekta (linije) in meje drugega geometrijskega objekta (zbirke točk) je –1, saj je njun presek prazen.

Tabela odnosa *Disjoint* med danima geometrijskima objektoma je torej:

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	−1	−1	1
Meja (A)	−1	−1	0
Zunanjost (A)	0	1	2

Ugotovili smo, da sta dana geometrijska objekta v odnosu *Disjoint*, saj njuna tabela ustreza vsem predpisanim pogojem.

2.1.2 DOTIK GEOMETRIJSKIH OBJEKTOV

Odnos *Touches* med dvema geometrijskima objektoma A in B lahko nastopi med pari *Polygon/LineString, Polygon/Point, LineString/Point, Polygon/Polygon, LineString/LineString.* Za dana geometrijska objekta A in B preverimo ali se dotikata tako, da pogledamo, ali velja
$(I(A) \cap I(B) = \emptyset) \wedge (A \cap B) \neq \emptyset$.

To pomeni, da

- se notranjosti geometrijskih objektov ne smejo sekati;
- roba geometrijskih objektov A in B se sekata.

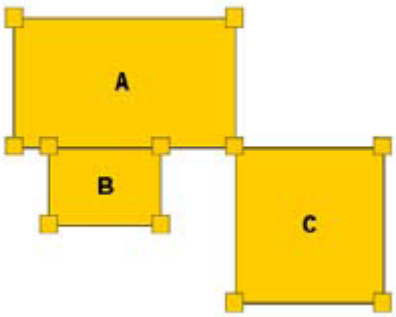
Da sta dana geometrijska objekta v odnosu *Touches*, mora veljati vsaj ena od tabel:

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	F	T	*
Meja (A)	*	*	*
Zunanjost (A)	*	*	*

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	F	*	*
Meja (A)	T	*	*
Zunanjost (A)	*	*	*

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	F	*	*
Meja (A)	*	T	*
Zunanjost (A)	*	*	*

Poglejmo si primer na Slika 16, kjer so A, B in C poligoni. Ker se A in B dotikata, mora veljati $(I(A) \cap I(B) = \emptyset) \wedge (A \cap B) \neq \emptyset$. To pomeni, da se notranjosti geometrijskih objektov ne smeta sekati, rob geometrijskega objekta A pa lahko seka notranjost geometrijskega objekta B ali obratno.



Slika 16 Odnos Touches (http://www.gis.hsr.ch/wiki/images/3/3d/9dem_springer.pdf)

Oglejmo si, kako bi izpolnili celotno tabelo. Ker se notranjosti geometrijskih objektov A in B ne sekata, je dimenzija preseka enaka –1 (–1 označuje dimenzijo nedefiniranega geometrijskega objekta). Dimenzija preseka meje geometrijskega objekta A in meje geometrijskega objekta B je 1. Njun presek je namreč linija, ki je enodimenzionalni geometrijski objekt. Dimenzija preseka notranjosti geometrijskega objekta A in zunanjosti geometrijskega objekta B je 2, ker je njun presek poligon, ki ima dimenzijo 2.

Tabela odnosa med danima geometrijskima objektoma je torej:

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	−1	−1	2
Meja (A)	−1	1	1
Zunanjost (A)	2	1	2

Ugotovili smo, da sta dana geometrijska objekta v odnosu *Touches*, saj njuna tabela ustreza tretjemu med navedenimi vzorci tabel za ta odnos.

2.1.3 KRIŽANJE GEOMETRIJSKIH OBJEKTOV

Odnos *Crosses* med dvema geometrijskima objektoma A in B lahko nastopi med pari *Point/Line*, *Point/Polygon*, *Line/Line in Polygon/Line*. Za dana geometrijska objekta A in B preverimo, ali se križata, tako da pogledamo, ali velja
$(\dim(I(A) \cap I(B) < \max(\dim(I(A)), \dim(I(B))))) \wedge (A \cap B \neq A) \wedge (A \cap B \neq B)$.

To pomeni:

- da je dimenzija preseka notranjosti obeh geometrijskih objektov manjša od dimenzije notranjosti vsaj enega geometrijskega objekta, ki ju primerjamo;
- noben geometrijski objekt ne sme biti v celoti vsebovan v drugem geometrijskem objektu.

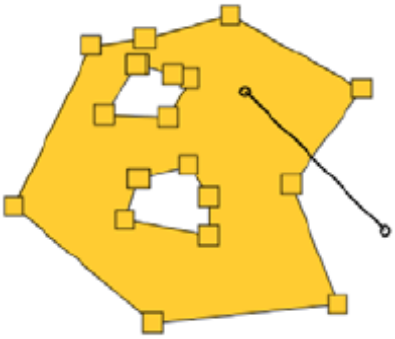
Da sta dana geometrijska objekta v odnosu *Crosses*, mora veljati bodisi tabela:

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	T	*	T
Meja (A)	*	*	*
Zunanjost (A)	*	*	*

bodisi:

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	0	*	*
Meja (A)	*	*	*
Zunanjost (A)	*	*	*

Poglejmo si primer na Slika 17, kjer je A poligon in B linija. Geometrijski objekt B ni v celoti vsebovan v geometrijskem objektu A. Dimenzija preseka notranjosti teh dveh geometrijskih objektov je manjša od dimenzije poligona, ki ima vrednost 2. Zato je dimenzija preseka notranjosti obeh geometrijskih objektov res manjša od večje dimenzije geometrijskih objektov ($1 < 2$). Torej omenjeni pogoji $(\dim(I(A) \cap I(B) < \max(\dim(I(A)), \dim(I(B))))) \wedge (A \cap B \neq A) \wedge (A \cap B \neq B)$ veljajo. Zato rečemo, da se objekta A in B križata (sta v odnosu *Crosses*).



Slika 17 Odnos Crosses (http://www.gis.hsr.ch/wiki/images/3/3d/9dem_springer.pdf)

Oglejmo si, kaj pove tabela odnosov. Ker se notranjosti geometrijskih objektov A in B sekata, je dimenzija preseka enaka 1 (1 označuje dimenzijo enodimenzionalnega geometrijskega objekta – linija). Dimenzija preseka notranjosti geometrijskega objekta A in meje geometrijskega objekta B je 0, ker je njun presek točka. Dimenzija preseka notranjosti geometrijskega objekta A in zunanjosti geometrijskega objekta B je 2, saj je njun presek dvodimenzionalen geometrijski objekt. Dimenzija preseka meje geometrijskega objekta A in meje geometrijskega objekta B je −1, saj preseka ni.

Tabela odnosa med danima geometrijskima objektoma je torej:

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	1	0	2
Meja (A)	0	−1	1
Zunanjost (A)	1	0	2

Za odnos *Crosses* sta pomembni le osenčeni polji. Ker sta obe različni od -1, torej ustrezata prvi vzorčni tabeli tega odnosa. S tem smo potrdili, da se objekta res križata.

2.1.4 OBJEKT LEŽI ZNOTRAJ DRUGEGA OBJEKTA

Objekta sta v odnosu *Within*, če eden leži znotraj drugega. V takem odnosu so lahko pari *Polygon/Polygon, Polygon/LineString, LineString/LineString, Polygon/Point*. Ali dani geometrijski objekt leži znotraj drugega geometrijskega objekta, preverimo tako, da pogledamo veljavnost izraza $(A \cap B = A) \wedge (I(A) \cap E(B) = \emptyset$.

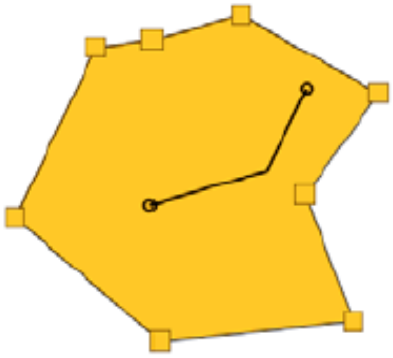
Izraz je veljaven, kadar je:

- geometrijski objekt v celoti vsebovan v drugem geometrijskem objektu;
- presek notranjosti prvega objekta z zunanjostjo drugega objekta prazen.

Da sta dana geometrijska objekta v odnosu *Within*, morajo veljati vrednosti v tabeli:

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	T	*	F
Meja (A)	*	*	F
Zunanjost (A)	*	*	*

Poglejmo si primer na Slika 18, kjer je A poligon in B linija. Geometrijski objekt B je v celoti vsebovan v objektu A. Presek notranjosti objekta A z zunanjostjo objekta B je prazen. To pomeni, da prvi geometrijski objekt ne seka zunanjosti drugega objekta. Torej omenjena pogoja $(A \cap B = A)$ in $(I(A) \cap E(B) = \emptyset$ veljata. Zato rečemo, da sta objekta A in B v odnosu *Within*.



Slika 18 Odnos Within (http://www.gis.hsr.ch/wiki/images/3/3d/9dem_springer.pdf)

Oglejmo si še, kako bi izpolnili celotno tabelo. Ker je geometrijski objekt B – linija v celoti vsebovan v geometrijskem objektu A – poligon, je dimenzija preseka 1, objekt z dimenzijo 1, saj je njun presek linija. Presek notranjosti objekta A in zunanjosti objekta B je prazen, zato je dimenzija preseka -1. Presek meje objekta A in zunanjosti objekta B je prazen, zato je dimenzija preseka -1.

Tabela za dana geometrijska objekta je torej:

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	1	-1	-1
Meja (A)	0	-1	-1
Zunanjost (A)	2	1	2

Ugotovili smo, da sta dana geometrijska objekta v odnosu *Within*, saj njuna tabela ustreza vsem predpisanim pogojem.

2.1.5 PREKRIVANJE GEOMETRIJSKIH OBJEKTOV

Odnos *Overlaps* med dvema geometrijskima objektoma A in B lahko nastopi med pari *Polygon/Polygon*, *LineString/LineString*, *Point/Point*. Ali dani geometrijski objekt prekriva drugi geometrijski objekt, preverimo tako, da pogledamo veljavnost izraza
$(\dim(\mathrm{I}(\mathrm{A})) = \dim(\mathrm{I}(\mathrm{B})) = \dim(\mathrm{I}(\mathrm{A}) \cap \mathrm{I}(\mathrm{B}))) \wedge (\mathrm{A} \cap \mathrm{B} \neq \mathrm{A}) \wedge (\mathrm{A} \cap \mathrm{B} \neq \mathrm{B})$.

Izraz je veljaven, kadar:

- je dimenzija notranjosti prvega geometrijskega objekta enaka dimenziji notranjosti drugega geometrijskega objekta;
- je dimenzija preseka notranjosti obeh geometrijskih objektov enaka dimenziji notranjosti obeh geometrijskih objektov;
- presek obeh objektov ne sme biti enak geometrijskima objektoma.

Da sta dana geometrijska objekta v odnosu *Overlaps*, mora veljati vsaj ena od tabel:

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	T	*	T
Meja (A)	*	*	*
Zunanjost (A)	T	*	*

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	1	*	T
Meja (A)	*	*	*
Zunanjost (A)	T	*	*

Poglejmo si primer na Slika 19, kjer sta A in B liniji. Dimenzija notranjosti objekta A je enaka dimenziji notranjosti objekta B, saj je dimenzija notranjosti objektov enaka 1. Njun presek je linija, ki je enodimenzionalen geometrijski objekt in je drugačne oblike kot sta dana geometrijska objekta. Ker je njun presek notranjosti linija, velja tudi, da je presek enak dimenziji notranjosti obeh geometrijskih objektov.

Torej omenjeni pogoji $\dim(\mathrm{I}(\mathrm{A})) = \dim(\mathrm{I}(\mathrm{B})) = \dim(\mathrm{I}(\mathrm{A}) \cap \mathrm{I}(\mathrm{B})) = 1$ in $(\mathrm{A} \cap \mathrm{B} \neq \mathrm{A})$, $(\mathrm{A} \cap \mathrm{B} \neq \mathrm{B})$ veljajo. Zato rečemo, da se objekta A in B prekrivata (sta v odnosu *Overlaps*).

Slika 19 Odnos Overlaps (http://www.gis.hsr.ch/wiki/images/3/3d/9dem_springer.pdf)

Oglejmo si še, kako bi izpolnili celotno tabelo. Dimenzija preseka notranjosti geometrijskega objekta A in notranjosti geometrijskega objekta B je 1, saj je njun presek notranjosti linija, objekt z dimenzijo 1. Dimenzija preseka zunanjosti geometrijskih objektov A in B je 2, saj je njun presek kar dvodimenzionalen geometrijski objekt. Dimenzija preseka zunanjosti objekta A in notranjosti objekta B je 1, saj je njun presek linija, objekt z dimenzijo 1.

Tabela odnosa med danima geometrijskima objektoma je torej:

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	1	−1	1
Meja (A)	−1	−1	0
Zunanjost (A)	1	0	2

Ugotovili smo, da sta dana geometrijska objekta v odnosu *Overlaps*, saj njuna tabela ustreza drugemu med navedenimi vzorci tabel za ta odnos.

2.1.6 SEKANJE GEOMETRIJSKIH OBJEKTOV

Odnos *Intersect* je inverzen odnosu *Disjoint*. Takrat velja $A \cap B \neq \emptyset$. Presek dveh sekajočih se geometrijskih objektov torej ne sme biti prazen.

Da sta dana geometrijska objekta v odnosu *Intersect*, mora veljati vsaj ena od tabel:

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	T	*	*
Meja (A)	*	*	*
Zunanjost (A)	*	*	*

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	*	T	*
Meja (A)	*	*	*
Zunanjost (A)	*	*	*

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	*	*	*
Meja (A)	T	*	*
Zunanjost (A)	*	*	*

	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	*	*	*
Meja (A)	*	T	*
Zunanjost (A)	*	*	*

Poglejmo si primer na Slika 20, kjer sta A in B liniji. Presek teh geometrijskih objektov je točka. Torej omenjeni pogoj $A \cap B \neq \varnothing$ velja. Zato rečemo, da se objekta A in B sekata (sta v odnosu *Intersect*).

Slika 20 Odnos Intersects (http://www.gis.hsr.ch/wiki/images/3/3d/9dem_springer.pdf)

Oglejmo si še, kako bi izpolnili celotno tabelo. Ker se objekta sekata, je dimenzija preseka notranjosti geometrijskih objektov 0 (0 označuje objekt brez dimenzije). Ugotovili smo, da sta dana geometrijska objekta v odnosu *Overlaps*, saj njuna tabela ustreza prvemu med navedenimi vzorci tabel za ta odnos.

Poglejmo si še celotno rešitev tabele tega odnosa. In sicer, dimenzija preseka notranjosti prvega geometrijskega objekta in meje drugega geometrijskega objekta −1 (−1 označuje dimenzijo nedefiniranega geometrijskega objekta). Dimenzija preseka notranjosti prvega geometrijskega objekta in zunanjosti drugega geometrijskega objekta je 1 (označuje dimenzijo enodimenzionalnega geometrijskega objekta – linija). Dimenzija preseka meje prvega geometrijskega objekta in notranjosti drugega geometrijskega objekta je −1 (−1 označuje dimenzijo nedefiniranega geometrijskega objekta). Dimenzija preseka mej obeh danih geometrijskih objektov je −1 (−1 označuje dimenzijo nedefiniranega geometrijskega objekta). Dimenzija preseka meje prvega geometrijskega objekta in zunanjosti drugega geometrijskega objekta je 0 (označuje dimenzijo brezdimenzionalnega geometrijskega objekta – točka). Dimenzija preseka zunanjosti prvega geometrijskega objekta in notranjosti drugega geometrijskega objekta je 1, ker je njun presek linija, ki ima dimenzijo 1. Dimenzija preseka zunanjosti prvega geometrijskega objekta in meje drugega geometrijskega objekta je 0. Dimenzija preseka zunanjosti prvega geometrijskega objekta in zunanjosti drugega geometrijskega objekta je 2.

Tabela odnosov med tema dvema geometrijskima objektoma je torej:

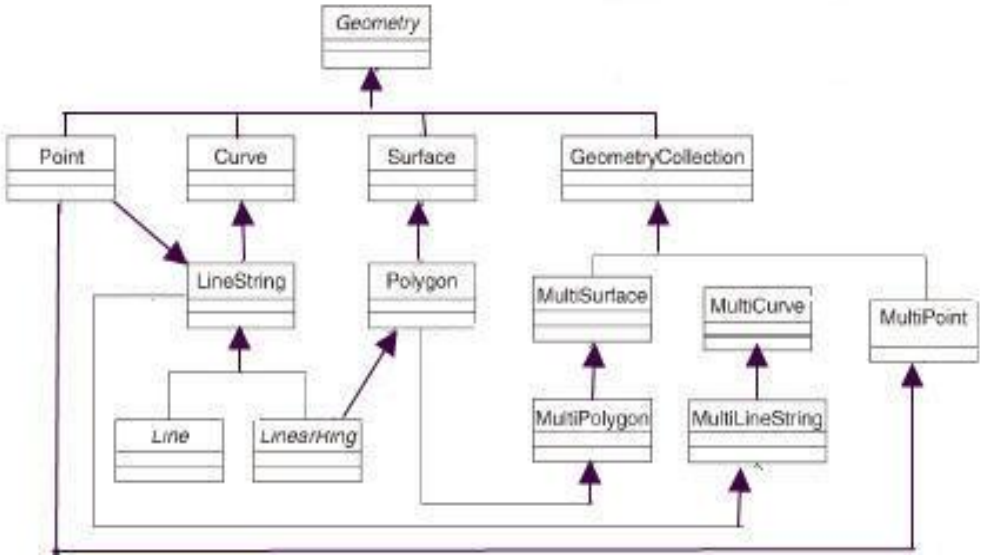
	Notranjost (B)	Meja (B)	Zunanjost(B)
Notranjost (A)	0	−1	1
Meja (A)	−1	−1	0
Zunanjost (A)	1	0	2

Za odnos *Intersect* je pomembno le osenčeno polje. Ker je polje enako 0, torej ustrezata vzorčni tabeli tega odnosa. S tem smo potrdili, da sta objekta res v odnosu *Intersect*.

2.2 GEOMETRIJSKI OBJEKTNI MODEL

V nadaljevanju si bomo pogledali zgradbo geometrijsko objektnega modela, kot ga je definiral OpenGIS Conzorcium (OGC).

Na Slika 21 je prikazana hierarhija razreda **Geometry**. Osnovana je na dimenzionalnem razširjenem modelu. V osnovi je to dvodimenzionalni model. Geometrijski objekti so torej podani s koordinatami x in y. Osnovni geometrijski razred je abstraktni razred **Geometry** in ima podrazrede **Point** (točka), **Curve** (krivulja), **Surface** (ploskev) in **GeometryCollection** (geometrijska zbirka). Vsak geometrijski objekt ponazarja ustrezen dejanski model geografskega objekta.



Slika 21 Hierarhija razreda Geometry (http://postgis.refractions.net/download/postgis-1.3.3.pdf)

Poglejmo si hierarhijo razreda **Geometry**, kot prikazuje Slika 21:

- podrazred **Point** (točka) predstavlja brezdimenzionalne geometrijske objekte;
- podrazred **Curve** (krivulja) predstavlja enodimenzionalne geometrijske objekte in vsebuje le en podrazred **LineString**. Iz **LineString** pa sta izpeljana razreda **Line** in **LinearRing**;
- podrazred **Surface** predstavlja dvodimenzionalne geometrijske objekte in vsebuje podrazred **Polygon**. Iz razreda **Polygon** pa je izpeljan razred **LinearRing**;
- podrazred **GeometryCollection** ima nič-, eno- in dvodimenzionalno zbirko razredov, kot so **MultiPoint**, **MultiLineString**, **MultiCurve**, **MultiSurface** in **MultiPolygon**, ti razredi pa ustrezajo zbirkam razredom **Point**, **LineString**, **Curve**, **Surface** *in* **Polygon**;

V nadaljevanju si poglejmo opis geometrijskih razredov, ki sestavljajo geometrijski objektni model. Za vsak geometrijski razred je opisanih le nekaj najznačilnejših metod in lastnosti.

2.2.1 RAZRED GEOMETRY

Je abstraktni razred za predstavitev nič-, eno- in dvodimenzionalnih geometrijskih objektov. Kot tak določa metode, ki jih morajo vsebovati vsi razredi v tem modelu. Določa osnovne metode na geometrijskih objektih. Te so:

- **GeometryType()**, ki nam vrne tip geometrijskega objekta;
- **SRID()**, ki nam vrne `ID` (identifikacijski podatek) geografskega koordinatnega sistema za geometrijski objekt. `SRID` ali geografski referenčni identifikator določa povezavo geometrijskih objektov z željenim koordinatnim sistemom.

Razred **Geometry** opredeljuje tudi metode za preverjanje geografskih odnosov med geometrijskimi objekti. Te kot rezultat vrnejo vrednost tipa **boolean** (*true* ali *false*). Metode so realizacija odnosov med objekti, opisanimi v razdelku 2.1. Te metode so:

- **Equals**, s katero preverimo, ali sta dana geometrijska objekta enaka;
- **Disjoint**, s katero preverimo, ali sta dana geometrijska objekta disjunktna;
- **Intersects**, s katero preverimo, ali se dana geometrijska objekta sekata;
- **Touches**, s katero preverimo, ali se dana geometrijska objekta dotikata;
- **Crosses**, s katero preverimo, ali se dana geometrijska objekta križata;
- **Within**, s katero preverimo, ali je geometrijski objekt vsebovan v drugem geometrijskem objektu;
- **Overlaps**, s katero preverimo, ali geometrijski objekt prekriva drugi geometrijski objekt.

Razred **Geometry**, določa tudi metode, ki podpirajo geografsko analizo med geometrijskimi objekti. To so:

- **Distance**, ki nam vrne najkrajšo razdaljo med dvema geometrijskima objektoma;
- **Intersection**, ki nam vrne geometrijski objekt, ki predstavlja presek dveh geometrijskih objektov;
- **Union**, ki nam vrne geometrijski objekt, ki predstavlja unijo dveh geometrijskih objektov;

Na primeru (glej stran 48) si poglejmo uporabo metode **GeometryType**. Uporabimo jo v funkciji `izvozSHPizPG`, ki jo potrebujemo za izvoz datotek SHP iz PostgreSQL-a. Tam smo napisali nekaj podobnega kot:

```
"SELECT count(*), geometrytype(the_geom) FROM" + tabela + " GROUP BY
geometrytype(the_geom);"
```

Ta ukaz nam vrne število zapisov, ki pripadajo določenemu geometrijskemu tipu, skratka, prešteje nam, koliko je v tabeli `tabela` točk ali linij ali poligonov, torej tistega tipa, ki je zapisan v stolpcu `the_geom`. Če zgornji ukaz v jeziku SQL uporabimo, vrne npr.:

```
count | geometrytype
------+--------------
  859 | MULTIPOLYGON
```

V tabeli katere ime smo shranili v spremenljivko `tabela` smo imeli torej izključno poligone.
Poglejmo si še en primer uporabe metode **GeometryType**, ki jo tudi uporabimo v funkciji `izvozSHPizPG`. Tam smo napisali:

```
"DELETE FROM " + tabela + " WHERE geometrytype(the_geom) <> '" + geomTip +
"'  AND geometrytype(the_geom) <> 'MULTI" + geomTip + "';"
```

Imamo spremenljivko `geomTip`, v kateri so shranjeni le osnovni geometrijski tipi podatkov, ti so **Point**, **LineString** in **Polygon**. Če ima spremenljivka `geomTip` vrednost **Polygon** je naš ukaz :

```
"DELETE FROM " + tabela + " WHERE geometrytype(the_geom) <> 'POLYGON' AND
geometrytype(the_geom) <> 'MULTIPOLYGON';"
```

S tem ukazom želimo v tabeli `tabela` obdržati le en geometrijski tip, bodisi kot enostaven (npr. **Polygon**) ali kot zbirko (**MultiPolygon**). Če so v tabeli `tabela` poleg poligonov na primer še točke, ki so tipa **Point**, se te v tabeli, katere ime je v `tabela`, izbrišejo.

2.2.2 RAZRED POINT (Točka)

Razred **Point** je podrazred razreda **Geometry**. Uporabimo ga za predstavitev geometrijskih objektov, ki nimajo dimenzije, torej točk. Najpomembnejši metodi tega razreda sta metodi, ki vrneta koordinatni vrednosti točke. To sta metodi:

- **X( )**, ki nam vrne x-koordinatno vrednost danega geometrijskega objekta točka;
- **Y( )**, ki nam vrne y-koordinatno vrednost danega geometrijskega objekta točka.

Poglejmo si uporabo teh metod še na primeru. Dano imamo tabelo **stavbe**, ki vsebuje podatke o geometriji stavb. Ti podatki so shranjeni v stolpcu `the_geom` tipa `geometry`. Zanimajo nas podatki o položaju stavb oz. njihove x in y koordinate. Izpišimo najprej podatke v tem stolpcu. Zaradi enostavnosti bomo prikazali le eno vrstico tabele.

```
"SELECT the_geom FROM stavbe limit 1;"

the_geom
----------------------------------------
0102002220FKL3D20000020004400046F8EDA…
```

Vidimo, da so podatki o položaju geometrijskih objektov v tabeli shranjeni kodirano. Zato nam ne pomenijo kaj dosti. Da nam bodo podatki v `the_geom` bolj jasni, si pomagamo s funkcijo **AsText( )**, ki izpiše podatke o geometriji geometrijskega objekta v tekstovni obliki. Torej:

```
"SELECT the_geom, AsText(the_geom) AS besedni_opis_the_geom FROM stavbe
limit 1;"

the_geom                                  | besedni_opis_the_geom
------------------------------------------------------------------
0102002220FKL3D20000020004400046F8EDA… | POINT(242793, 4473294)
```

Sedaj pa si poglejmo uporabo metode **X( )** in **Y( )**, ki vrneta koordinati točke.

```
"SELECT X(the_geom) AS x_vrednost, Y(the_geom) AS y_vrednost FROM stavbe
limit 1"

x_vrednost | y_vrednost
-----------+-----------
242793     | 4473294
```

2.2.3 RAZRED CURVE(Krivulja), LINESTRING (Interpolacijska krivulja), LINE (Premica), LINEARRING (Zaprta krivulja)

Razred **Curve** je podrazred razreda **Geometry**. Je abstraktni razred, torej ne moremo narediti objektov tipa **Curve**. Služi zato, da definiramo določene metode, ki so skupne vsem krivuljam, torej enodimenzionalnim geometrijskim objektom.

Za tip **Curve** običajno uporabljamo predstavitev v obliki zaporedja točk. Glede na to, kakšna krivulja nastane z interpolacijo točk tega zaporedja, imamo na voljo ustrezne podrazrede. Trenutno se uporablja le linearna interpolacija. Zato imamo na voljo le en podrazred, in sicer **LineString**.

Oglejmo si določene definicije in lastnosti, povezane s krivuljami v tem geometrijskem modelu:

- krivulja je preprosta, če ne gre skozi nobeno točko dvakrat;
- krivulja je sklenjena, če je začetna točka enaka končni;
- rob sklenjene krivulje je prazen;
- rob nesklenjene krivulje sestavljata začetna in končna točka.

Definirajmo še nekaj metod razreda **Curve**:

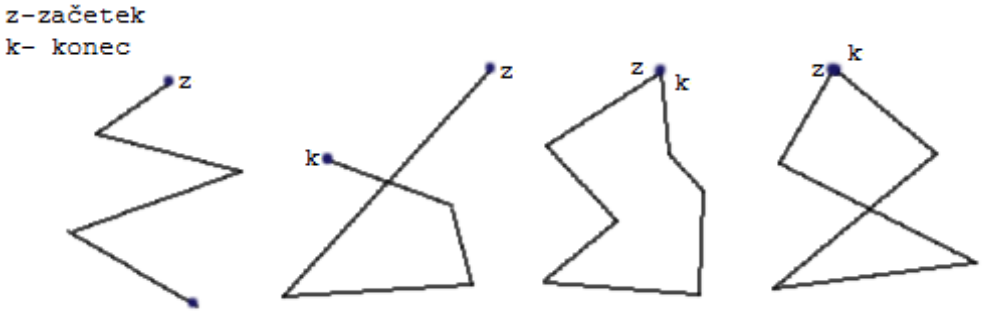
- **Length( )**, s katero izračunamo dolžino krivulje;
- **StartPoint( )**, s katero dobimo začetno točko krivulje;
- **EndPoint( )**, s katero dobimo končno točko krivulje;
- **IsClosed( )**, s katero preverimo, če je krivulja sklenjena;
- **IsRing( )**, s katero preverimo, če je krivulja sklenjena in preprosta.

Kot smo rekli, je iz razreda **Curve** trenutno izpeljan le razred **LineString**. Objekt tipa **LineString** je določen z zaporedjem točk. **LineString** je krivulja, ki jo dobimo z linearno interpolacijo med temi točkami. Vsak zaporedni par točk na interpolirani krivulji torej definira daljico. S pomočjo objektov tega razreda predstavimo na primer ceste, reke ...

Za krivuljo v razredu **LinearRing** mora veljati, da je krivulja preprosta in sklenjena. To opišemo z:
$(\forall x1, x2 \in (a, b] x1 \neq x2 \Rightarrow f(x1) \neq f(x2)) \wedge (\forall x1, x2 \in [a, b) x1 \neq x2 \Rightarrow f(x1) \neq f(x2))$.
Kar pomeni, da se krivulja ne sme sekati, pri tem pa sta začetna in končna točka lahko enaki ali različni.

Poglejmo si primere krivulj tipa **LineString (**Slika 22).

- Prva in tretja krivulji sta preprosti. Razlika med njima je v tem, da je tretja krivulja tudi sklenjena;
- Druga in četrta krivulji pa nista preprosti krivulji, hkrati pa je četrta krivulja še sklenjena. Seveda to velja pri predpostavki, da stvar gledamo ravninsko. Če pa je omenjena slika le dvodimenzionalni prikaz krivulj, ki ležijo v prostoru, sta seveda tudi druga in četrta krivulja lahko preprosti (točka "sekanja" sta pravzaprav dve toči z različnima z koordinatama).



Slika 22 Primeri krivulj tipa LineString

Če imamo v zaporedju točno dve točki, dobimo podrazred **Line**. Če pa je krivulja tipa **LineString** tako sklenjena kot tudi preprosta, dobimo enodimenzionalni geometrijski objekt iz podrazreda **LinearRing**. Iz razreda **LineString** sta tako izpeljana razreda **Line** in **LinearRing**. Objekt razreda **Line** torej opisuje daljico (linijo). Objekt tipa **LinearRing** (obroč) predstavlja sklenjeno zaporedje točk, kjer se v vsaki točki srečata točno dve liniji.

Sedaj si poglejmo še nekaj primerov uporabe zgoraj navedenih metod. Dano imamo tabelo **ceste**, ki vsebuje podatke o geometriji cest. Ti podatki so shranjeni v stolpcu `the_geom` tipa `geometry`. Radi bi izračunali dolžine cest. Zato bomo uporabili metodo **Length( )**. Kot že vemo, je cesta v GIS-u predstavljena kot linija, zato je geometrijskega tipa **LineString**. Izpišimo najprej podatke v stolpcu `the_geom`. Zaradi enostavnosti bomo prikazali le eno vrstico tabele. Ker vemo, da so podatki v stolpcu kodirani, bomo s pomočjo metode **AsText( )** podatke izpisali v tekstovni obliki.

```
"SELECT the_geom, AsText(the_geom) AS besedni_opis_the_geom FROM ceste
limit 1;"

the_geom                       | besedni_opis_the_geom
-------------------------------------------------------------------
01050000020E6100000001012…  | LINESTRING(279 3678,409 543,656 512,739 876)
```

Zgornji zgled nam je pokazal, kakšne podatke imamo v tabeli. Ker vemo, da so podatki dani v metrih, moramo ustrezno dolžino še deliti s 1000, da bomo dobili dolžino v kilometrih. Zato napišemo

```
"SELECT (Length(the_geom)/1000) AS dolzina_ceste_km FROM ceste limit 1;"

dolzina_ceste_km
------------------
2.8284271247462
```

Ugotovili smo torej, da je v tabeli **ceste** cesta dolga slabe 3 kilometre.

2.2.4 RAZRED SURFACE (Ploskev), POLYGON (Poligon ali večkotnik)

Razred **Surface** je podrazred **Geometry** razreda. Je abstraktni razred, torej ne moremo narediti objektov tipa **Surface**. Služi zato, da definiramo določene metode, ki so skupne vsem ploskvam, torej dvodimenzionalnim geometrijskim objektom.

Objekt tega razreda je preprosta ploskev, ki je sestavljena iz ene »krpe«, ki ima eno zunanjo mejo in nič ali več notranjih meja. To pomeni, da je ploskev sklenjena in preprosta in predstavlja geometrijski objekt **Polygon**. Notranjost ploskve je razdeljena na trikotnike, ki se stikajo na ogliščih poligona. Pri tem upoštevamo še prekrivanje trikotnikov, ki so v notranjosti ploskve. O prekrivanju geometrijskih objektov smo govorili v poglavju 2.1.5 v katerem je opisan prostorski odnos *Overlaps*, s katerim preverimo ali dani geometrijski objekti (trikotniki) prekrivajo drugi geometrijski objekt (ploskev).

Pomembnejši metodi, ki jih vsebuje razred **Surface**, sta:

- **Area( )**, ki nam vrne površino dane ploskve;
- **Centroid( )**, ki nam vrne težiščno točko dane ploske;

Razred **Surface** vsebuje le razred **Polygon**. Objekt tipa **Polygon** je določen kot sklenjeno zaporedje točk (oglišč) in daljic (stranic), tako da se po dve stranici stikata v oglišču. Notranjost poligona je preprosta ploskev, ki ima ploščino. Za poligone je tudi značilno, da se stranice ne sekajo. Vemo, da objekt **LinearRing** določa sklenjeno zaporedje točk, kjer se v vsaki točki srečata točno dve liniji. Torej je iz razreda **LinearRing** izpeljan le razred **Polygon**.

Poglejmo si še nekaj lastnosti, ki veljajo za poligone:

- poligoni so topološko zaprti;
- meja poligona je sestavljena iz **LinearRing**, ki določajo poligonu zunanjo in notranjo mejo;
- objekti tipa **LinearRing** (obroči) morajo imeti presek mej prazen, vendar se lahko sekajo v meji s poligonom le v točki;
- notranjost poligona je povezana zbirka točk;
- zunanjost poligona z eno ali več objektom tipa **LinearRing** ni povezana.

Metode, ki jih vsebuje razred **Polygon**, so:

- **ExteriorRing( )**, ki nam vrne rob poligona. Rob poligona je geometrijski objekt tipa **LineString**;
- **NumExteriorRing( )**, ki nam vrne število objektov tipa **LinearRing** v poligonu;
- **NumInteriorRing( )** nam vrne število objektov tipa **LinearRing** v poligonu.

Sedaj si poglejmo še nekaj primerov uporabe zgoraj navedenih metod. Dano imamo tabelo **drzava**, ki vsebuje podatke o geometriji določenih držav. Ti podatki so shranjeni v stolpcu `the_geom` tipa `geometry`. Radi bi izračunali površino države. Zato bomo uporabili metodo **Area( )**. Kot vemo, je država predstavljena kot poligon, zato je geometrijskega tipa **Polygon**. Izpišimo najprej podatke v stolpcu `the_geom`. Zaradi enostavnosti bomo prikazali le eno vrstico tabele.

```
"SELECT AsText(the_geom) AS besedni_opis_the_geom FROM drzava LIMIT 1;"

besedni_opis_the_geom
------------------------------------------------------------------------
POLYGON(2578934 35940,459034 545489,6583094 53483,7859034 858934 …)
```

Zgornji zgled nam je pokazal, kakšne podatke imamo v tabeli. Ker vemo, da so podatki dani v metrih, moramo ustrezno površino še deliti s 1000, da bomo dobili površino v kvadratnih kilometrih. Zato napišemo

```
"SELECT Area(the_geom)/1000 AS povrsina_km2 FROM drzava LIMIT 1;"

povrsina_km2
-------------
20,253
```

Ugotovili smo torej, da ima v tabeli **drzava** poršino okrog 20,253 km^2.

2.2.5 RAZRED GEOMETRYCOLLECTION (Geometrijska zbirka)

Pogosto je koristno, da imamo možnost združiti večje število geometrijskih objektov. Tako bi npr. lahko naredili zbirko, ki bi vsebovala vse ulice v nekem mestu, skupaj s točkami, ki bi predstavljale prometne znake na teh ulicah.

Razred **GeometryCollection** je podrazred razreda **Geometry** in je zbirka enega ali več geometrijskih objektov. Vsi objekti v zbirki morajo uporabljati isti koordinatni sistem. Možno pa je, da imamo v isti zbirki različne geometrijske objekte (na primer v isti zbirki so lahko linije in točke).
Ena izmed najbolj uporabnih metod razreda **GeometryCollection** je **NumGeometries( )**, ki vrne število geometrijskih objektov.

Poglejmo primer: dano imamo tabelo **ulice**, ki ima podatke o geometriji ulic in prometnih znakih. Ti podatki so shranjeni v stolpcu `the_geom` tipa `geometry`. Izpišimo najprej podatke v stolpcu `the_geom`. Zaradi enostavnosti bomo prikazali le dve vrstici tabele v tekstovni obliki.

```
"SELECT AsText(the_geom) AS besedni_opis_the_geom FROM ulice LIMIT 2;"

besedni_opis_the_geom
------------------------------------------------------------------
LINESTRING(24093 357590, 490534 55094, 64032 521902, 74802 87430)
POINT(24093, 357590)
```

Zgornji zgled nam je pokazal, kakšne podatke imamo v tabeli. Radi bi preverili, koliko geometrijskih objektov je v tabeli **ulice**. Pomagamo si z metodo razreda **GeometryCollection NumGeometries( )**, ki nam vrne število geometrijskih objektov.

```
"SELECT NumGeometries(the_geom) AS st_geom_objektov FROM ulice;"

st_geom_objektov
-----------------
11035465
```

Ugotovili smo torej, da so v tabeli **ulice** geometrijski objekti, in sicer objekti tipa LineString in objekti tipa **Point**. Skupni število geometrijskih objektov v tabeli **ulice** je 11035465.

Če se omejimo in naredimo zbirko objektov istega tipa, dobimo ustrezne podrazrede. Ti razredi so **MultiPoint** (zbirka točk), **MultiPolygon** (zbirka poligonov), **MultiSurface** (zbirka ploskev), **MultiLineString** (zbirka linij) in **MultiCurve** (zbirka krivulj).

- **RAZRED MULTIPOINT** (zbirka točk) je zbirka brezdimenzionalnih geometrijskih objektov, ki nimajo identičnih koordinatnih vrednosti. Tako bi lahko naredili zbirko, ki bi vsebovale vse hiše v neki državi, ki so predstavljene kot točka. Razred **MultiPoint** je podrazred razreda **GeometryCollection**.

- **RAZREDA MULTILINESTRING** (zbirka linij) in **MULTICURVE** (zbirka krivulj) Razred **MultiCurve** je abstraktni razred, torej ne moremo narediti objektov tipa **MultiCurve**. **MultiCurve** je abstraktni razred iz katerega je izpeljan le razred **MultiLinestring**, ta pa je izpeljan iz razreda **LineString**. Ena izmed metod razreda **MultiCurve** je **Lenght( )**, ki vrne skupno dolžino vseh krivulj.

- **RAZREDA MULTIPOLYGON** (zbirka poligonov) in **MULTISURFACE** (zbirka ploskev) **MultiSurface** (zbirka ploskev) je zbirka dvodimenzionalnih geometrijskih objektov, katere elementi so ploskve in je podrazred razreda **GeometryCollection**. **MultiSurface** je abstraktni razred in iz njega je izpeljan le razred **MultiPolygon**. Razred **MultiPolygon** abstraktni razred in je izpeljan iz razreda **Polygon** Ena izmed metod razreda **MultiSurface** je **Area( )**, ki nam vrne skupno površino vseh ploskev v zbirki.

3 DELO Z GEOGRAFSKIMI PODATKI IN POSTGIS-OM

V tem poglavju si bomo ogledali delo z geografskimi podatki. V poglavju 2 smo povedali, da geografske podatke običajno hranimo v relacijski bazi. V poglavju o GIS orodjih pa smo omenili, da določena GIS orodja, predvsem orodja za grafični prikaz geografskih podatkov, ne uporabljajo baze, ampak datoteke, oz. da uporabljajo točno določeno bazo v formatu DBF, ki pa nam ne ustreza. Zato si bomo v nadaljevanju pogledali, kako baze geografskih podatkov, ki so shranjene v formatu DBF, uvozimo v relacijsko bazo. Da si še malo osvežimo spomin, povejmo še enkrat, da v omenjenih orodjih za prikaz geografskega objekta potrebujemo tri vrste datotek. Te so SHP, DBF in SHX. Skupek teh datotek poimenujemo *shapefile* (**shape datoteka**).

Seveda lahko geografske podatke obdelamo, analiziramo … na več načinov. Pogledali si bomo, kako analizirati geografske podatke v relacijski bazi in kako analizirane ali obdelane podatke iz relacijske baze, ki jo uporabljamo s sistemom PostgreSQL, pretvorimo nazaj v **shape datoteko**.

3.1 UVOZ GEOGRAFSKIH PODATKOV V POSTGRESQL

Če želimo geografske podatke analizirati, nam PostGIS omogoča pretvarjanje geografskih podatkov iz **shape datotek** v obliko, v kateri te podatke vstavimo v relacijsko bazo, ki jo upravljamo s sistemom PostgreSQL. V ta namen vsebuje program **shp2pgsql.exe**, ki omogoča pretvarjanje **shape datotek** v SQL datoteke. Na teh datotekah so ukazi, ki omogočajo vnos podatkov v relacijsko bazo. Poglejmo si uporabo programa **shp2pgsql.exe** na primeru.

Dane imamo podatke o slovenskih cestah. Shranjeni so v **shape datotekah**. Datoteke, ki vsebujejo podatke o obliki Slovenije, so poimenovane z **drzava.***. Datoteke, ki vsebujejo podatke o slovenskih cestah, pa so poimenovane z **ceste.***. Radi bi shranili podatke o danih geografskih objektih v relacijsko bazo.

Poglejmo si, kakšne opcije nam omogoča program **shp2pgsql.exe**:

- –**d**, izprazni tabelo v relacijsko bazi, preden jo napolnimo s podatki;
- –**c**, uporabi SQL ukaz CREATE TABLE, ki ustvari tabelo;
- –**D**, uporabi SQL ukaz INSERT, ki omogoča vstavljanje podatkov v tabelo;
- **–s** <*SRID*>, s katerim navedemo identifikacijsko številko (`SRID` podatek) koordinatnega sistema v katerem so zapisani geografski podatki;
- **–W** <*encoding*>, pove, v katerem kodnem sistemu so podatki v datoteki DBF. V izhodni datoteki se uporabi kodiranje UTF-8.

Uporabimo lahko tudi parameter `-n` s katerim omogočimo uvoz podatkov tudi v primeru, če ni vseh treh datotek, ki so skupek poimenovan z **shape datoteka**, ampak imamo le datoteko DBF.

Ukaz, s katerim sestavimo skupek ukazov za uvoz geografskih podatkov za Slovenijo v relacijsko bazo, je oblike:

```
shp2pgsql -c -D -s 4326 drzava.shp uvoz_drzava -W ISO-10101 >
_tmp_drzava.sql
```

Podatki so shranjeni v **shape datoteki**, poimenovani **drzava.***. Ta ima podatke vnesene v kodni tabeli `ISO-10101`. Uporabljamo Lambertovo projekcijo, kar povemo s parametrom `-s` z vrednostjo 4326. S parametrom `-c` ustvarimo v relacijski bazi tabelo `uvoz_drzava` in jo zaradi parametra `-D` napolnimo z podatki, ki so shranjeni v **shape datoteki** (no, dejasnko sestavimo le ukaze vSQL, ki to naredijo, če jih izvedemo).

S tem ukazom, ki ga izvršimo v ukazni vrstici, torej sestavimo ustrezne ukaze v jeziku SQL in jih zapišemo v datoteko **tmp_drzava.sql**. Ukazi v tej datoteki nam omogočijo, da lastnosti geografski podatkov o Sloveniji vnesemo v relacijsko bazo.

Ker so v *shapefile* **drzava.*** podatki le o eni državi (Sloveniji), datoteka **_tmp_drzava.sql** vsebuje le en SQL ukaz INSERT. Poglejmo si vsebino te datoteke:

```
BEGIN;
CREATE TABLE "uvoz_drzava" (gid serial PRIMARY KEY, "id" bigint, "feattyp"
character varying (3), "featarea" bigint,  "name" character varying (70),
"namelc"  character varying (3), "munit smallint, the_geom geometry);
INSER INTO " uvoz_drzava " VALUES ('2', ' 17050083000001', '1111', 'SLV' ,
'20280226737', '1337638', 'Slovenija', 'SLV', '1',
'0106000020F4D2000001000000010300000001000000E36D0000F062EB75C7CA39414A…');
END;
```

Opazimo, da SQL datoteka vsebuje ukaz, s katerim ustvarimo tabelo **uvoz_drzava.** Ko je ta tabela narejena, se vanjo z ukazom INSERT vstavijo geografski podatki o obliki Slovenije. Te smo dobili iz *shapefile* **drzava.***.

Ukaz, s katerim pretvorimo geografske podatke o slovenskih cestah, ki so shranjeni v **shape datoteki**, poimenovanih z **ceste.***, v relacijsko bazo, je oblike:

```
shp2pgsql -c -D -s 4326 ceste.shp uvoz_ceste -W ISO-10101 > _tmp_ceste.sql
```

Tudi s tem ukazom sestavimo ustrezne ukaze v jeziku SQL in jih shranimo v datoteko **tmp_ceste.sql**. Ukazi v tej datoteki pa nam omogočijo, da lastnosti geografski podatkov o slovenskih cestah vnesemo v relacijsko bazo.

V *shapefile* **ceste.*** so shranjene vse slovenske ceste, zato datoteka **_tmp_ceste.sql** vsebuje veliko SQL ukazov INSERT. Zato si poglejmo le del vsebine datoteke **_tmp_ceste.sql**.

```
BEGIN;
CREATE TABLE "uvoz_ceste" (gid serial PRIMARY KEY,
"id" bigint, "feattyp" smallint, "ft" smallint, "f_jnctid" bigint,
"f_jncttyp" smallint, "t_jnctid" bigint, "t_jncttyp" smallint, "pj"
smallint, "meters" double precision, "frc" smallint, "netclass" smallint,
"netbclass" smallint, "net2class" smallint, "name" character varying (70),
"namelc" character varying (3), "sol" smallint, "nametyp" character varying
(2), "charge" character varying (2), "routenum" character varying (10),
"rtetyp" smallint, "rtedir" smallint, "rtedirvd" smallint, "procstat"
smallint, "fow" smallint, "sliprd" smallint, "freeway" smallint, "backrd"
smallint, "tollrd" smallint, "rdcond" smallint, "stubble" smallint,
"privaterd" smallint, "constatus" smallint, "oneway" smallint, "f_bp"
smallint, "t_bp" smallint, "f_elev" smallint, "kph" smallint, "munutes"
double precision, "posaccur" smallint, "carriage" smallint, "lanes"
smallint, "extentr" smallint, "laneval" smallint, "ramp" smallint,
"the_geom" geometry, "r_type" smallint, "r_itype" smallint, "r_rtesym"
character varying (4), "r_rteprio" smallint, "rtetyp2" smallint,
"routenum2" smallint);
INSERT INTO "uvoz_ceste" VALUES ('17051000271891', '4110', '0',
'100000000063208', '4', '100000000063192', '4', '0', '52.6', '4', '0', '0',
'2', 'Celovška cesta', 'SLV', '0', 'ON', '', '', '0', '', '', '1', '3',
'0', '0', '0', '', '1', '0', '0', '', '', '0', '0', '0', '0', '65',
'0.049', '0', '', '0', '0', '', '0',
'0102000020F4D200000200000046F8EDA…','4', '1', 'SRBW', '99', '0');
INSERT INTO "uvoz_ceste" VALUES ('17051001715825', '4110', '0',
'17050200107331', '0', '17050200106666', '0', '0', '39477', '7', '0', '0',
'5', 'Jadranska ulica, '', '0', '', '', '', '0', '', '', '2', '3', '0',
'0', '0', '', '3', '0', '0', '', '', '0', '0', '0', '0', '35', '0.052',
```

```
'0', '', '0', '0', '', '0', '0102000020F4D20000050000000F22…','', '0', '',
'', '');
INSERT INTO "uvoz_ceste" VALUES ('17051001715830', '4110', '0',
'11040200106589', '0', '17009346810645', '1', '1', '357', '1', '0', '0',
'3', 'Dunajska cesta', '', '0', '', '', '', '0', '', '', '4', '6', '1',
'0', '0', '', '3', '0', '0', '', '', '0', '0', '0', '0', '56', '0.2', '0',
'', '0', '0', '', '0', '0102000020F4D200000500000F090A…','', '0', '', '',
'');
…
END;
```

SQL datoteka vsebuje ukaz, ki ustvari tabelo **uvoz_ceste** in ukaze za polnjenje te tabele z geografskimi podatki o slovenskih cestah.

Klic programa **shp2pgsql** smo uporabili (glej stran 48) v funkciji `vnosSHPvPG`. Tam smo napisali nekaj podobnega kot:

```
var ukaz = psqlLokacija + "shp2pgsql -c -D -s 4326 " + shpDatoteka
   + " " + tabela;
if(parameter)
    ukaz += " " + parameter;
if(kodnaTabela)
   ukaz += " -W " + kodnaTabela;
ukaz += " > " + tempDatoteka;
```

V spremenljivko `ukaz` smo sestavili ustrezen klic programa. Ker so vsi podatki shranjeni v Lambertovi projekciji, se v ukazu `ukaz` vedno uporabi parameter `-s` z vrednostjo `4326`. V primeru, da geografski podatki niso shranjeni v obliki kodiranja UTF-8, uporabimo tudi parameter `kodnaTabela`, ki v `ukaz` doda parameter `-W` in vrednost kodne tabele.

3.2 IZVOZ GEOGRAFSKIH PODATKOV IZ POSTGRESQL

Če želimo obdelane ali analizirane geografske podatke, ki so shranjeni v PostgreSQL bazi pretvoriti v **shape datoteko**, nam tudi to omogoča PostGIS s programom **pgsql2shp.exe**. Poglejmo si uporabo progama **pgsql2shp.exe** na primeru.

V PostgreSQL bazi imamo dane geografske podatke o obliki Slovenije in o njenih cestah. Geografski podatki o Sloveniji so shranjeni v tabeli **uvoz_drzava**, podatki o cestah pa v **uvoz_ceste**. Geografske podatke, ki so shranjeni v tabelah, bi radi pretvorili v **shape datoteki.**

Program **pgsql2shp.exe** omogoča različne opcije:

- **–f** <*filename*> – zapiše podatke v izbrano datoteko;
- **–p** <*port*> – vrata za povezavo do relacijske baze;
- **–g** <*geometrijski stolpec*> – v primeru tabel, ki imajo več geometrijskih stolpcev (torej stolpcev z geometrijskimi podatki) je treba navesti, kateri geometrijski stolpec se uporabi pri tvorbi **shape datoteke**.

Poglejmo si primer uporabe programa **pgsql2shp**.

Denimo, da bi radi videli potek Jadranske ulice. V ta namen bomo uporabili program **Easy Map Explorer**. Ta potrebuje ustrezno **shape datoteko**. Zato moramo ustrezne podatke iz relacijske baze prenesti v tako datoteko. Ukaz, izvršen v ukazni vrstici, podatke o imenu in geometriji o Jadranski ulici, pridobljene iz tabele **uvoz_ceste,** shrani v **shape datoteko** imenovano **izvoz_ceste.***:

```
pgsql2shp -f  Z:\izhodnipodatki\izvoz_ceste.shp uvoz_ceste "SELECT name,
the_geom FROM uvoz_ceste WHERE name ='Jadranska ulica;'"
```

Izvozimo še podatke o imenu in geometriji Slovenije:

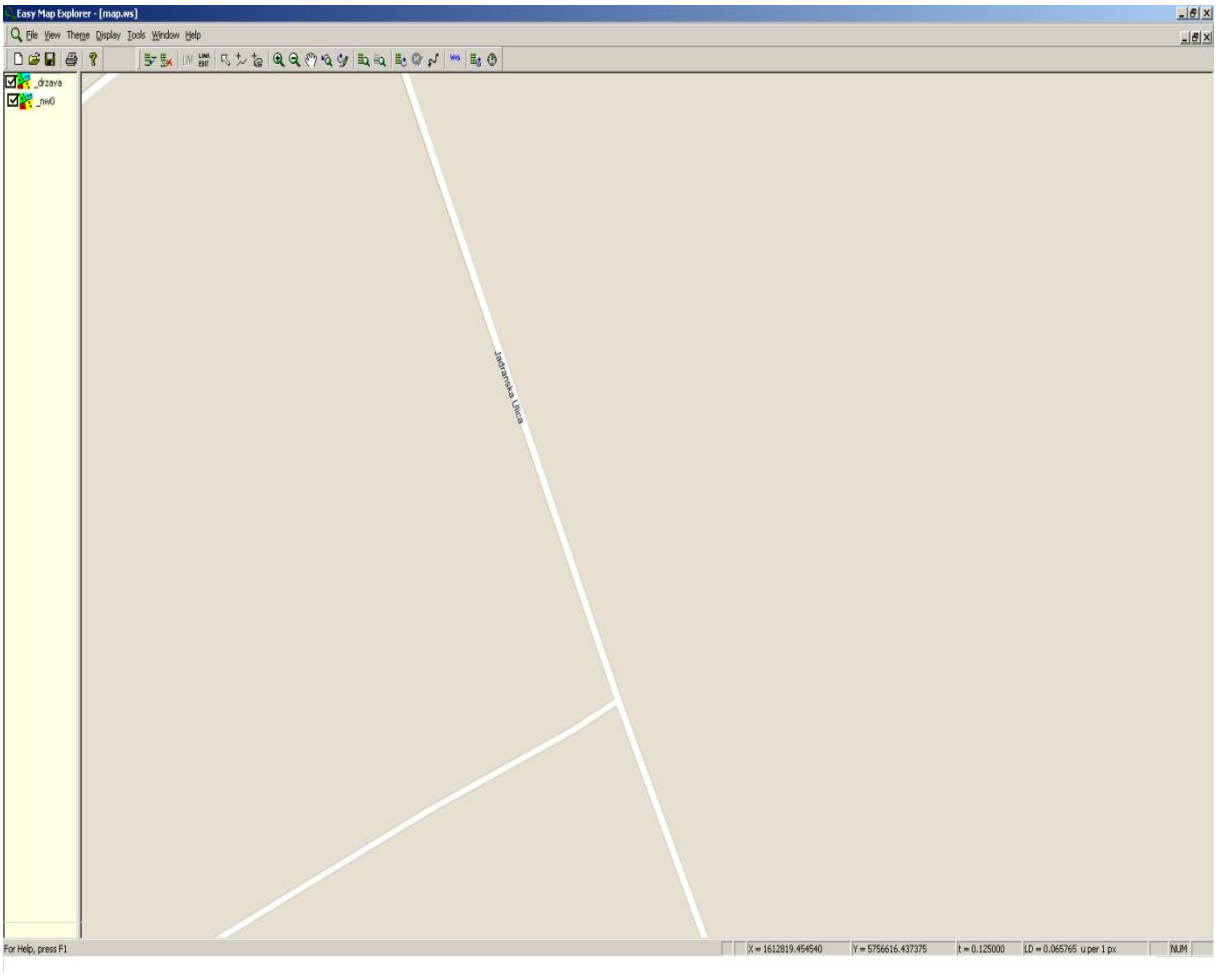
```
pgsql2shp -f  Z:\izhodnipodatki\izvoz_drzava.shp drzava "SELECT name,
the_geom FROM uvoz_drzava;'"
```

Program **pgsql2shp** smo (glej stran 48) uporabili v funkciji `izvoziShpDatoteke`. Tam smo napisali nekaj podobnega kot:

```
zagonUkaza(psqlLokacija + "pgsql2shp -u postgres -f " + shpDatoteka + " " +
database + " -p " + port + " " + sqlStavek);
```

Funkcija `zagonUkaza` nam v ukazni vrstici izvede ukaz, ki iz tabele v relacijski bazi izpiše podatke v **shape datoteko**. Spremenljivka `database` hrani podatke o imenu relacijske baze, iz katere se pretvorijo podatki v **shape datoteko**, katere ime je shranjeno v spremenljivki `shpDatoteka`. Ker pa je v relacijski bazi lahko več tabel, podatke pa lahko izvozimo le iz ene tabele, uporabimo SQL stavek, ki je shranjen v spremenljivki `sqlStavek`, s katerim določimo, iz katere tabele se pretvorijo podatki v **shape datoteko**.

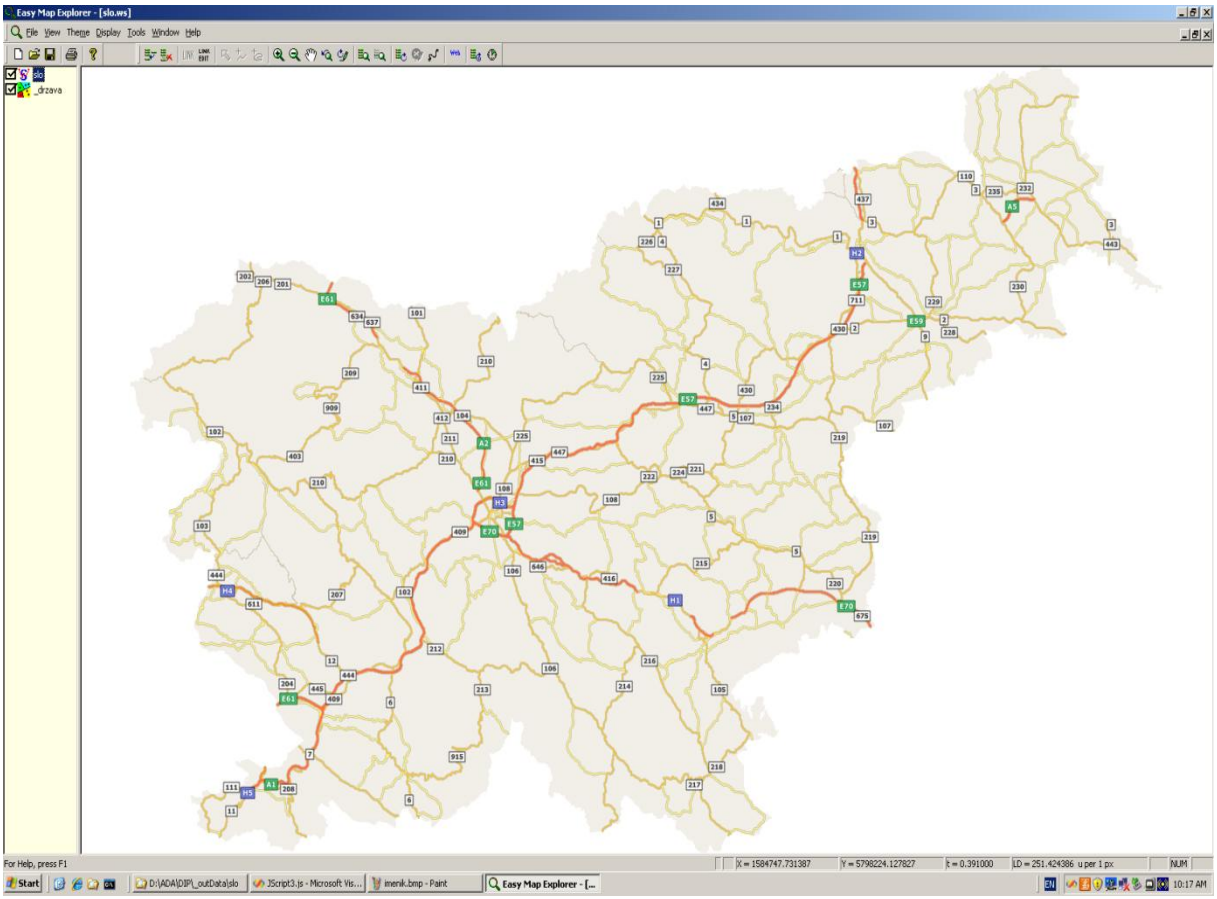
Sedaj imamo ustvarjeni novi datoteki, katerih vsebino lahko pogledamo v GIS orodju **Easy Map Explorer**. Na Slika 23 vidimo geografska objekta Slovenija in Jadranska ulica. Geografski objekt Slovenija je predstavljen kot poligon, Jadranska ulica pa kot linija.



Slika 23 Grafični prikaz geografskih objektov Jadranske ulice in Slovenije v GIS orodju

4 IZDELAVA KARTE SLOVENIJE S PROGRAMSKIM JEZIKOM JSCRIPT

Oglejmo si sedaj, kako s pomočjo opisanih tehnologij izdelamo karto (zemljevid). Kot programski jezik, ki bo omogočil povezavo vseh prej omenjenih tehnologij, bomo uporabili **JScript**. Kot zgled bom prikazala izdelavo karte Slovenije s cestami in njihovimi oznakami. Za grafični prikaz zemljevida bom uporabila GIS orodje **Easy Map Explorer** (Slika 24).



Slika 24 Grafični prikaz Slovenije z njenimi cestami in oznakami v GIS orodju

4.1 JSCRIPT

V času študija smo spoznali programska jezika Java in JavaScript.

Java je programski jezik, ki je neodvisen od platforme, na katerem teče. To pomeni, da je v neodvisen od kombinacije strojne opreme in operacijskega sistema. Za izvajanje prevedenega programa je zadolžen poseben modul, ki je srce vsakega z javo združljivega brskalnika ali operacijskega sistema. Java je objektni jezik.

JavaScript je skriptni jezik za spletne strani. Programe napisane v JavaScriptu zapišemo v HTML dokument in zato ne potrebuje nobenega posebnega prevajalnika. Za izvajanje skriptnega jezika poskrbi, tako kot tudi za HTML ukaze, brskalnik sam.

Čeprav sta si Java in JavaScript zelo podobna, je potrebno najprej opozoriti, da gre pri uporabi za precej ločeni tehniki programiranja. Java je programski jezik, medtem ko JavaScript skriptni jezik. Razlika je ta, da lahko z javo napišete program, ki se izvaja samostojno, program v JavaScriptu pa se izvaja izključno v kombinaciji z brskalnikom.

Microsoft je razvil še dva nova skriptna jezika, imenovana **VBScript** in **JScript**. **VBScript** je skriptni jezik, ki temelji na Microsoftovemu **Visual Basicu**. Razlika med JavaScriptom in VBScriptom, a ne velika, je v sintaksi jezika in v dodanih knjižnicah z gradniki.

JScript je še en skriptni jezik, s katerim se bomo ukvarjali v nadaljevanju. Je v bistvu JavaScript, ki pa ima dodane še nekatere gradnike. Torej, če povzamemo, JavaScript, VBScript in JScript so vsi skriptni jeziki, ki se le malenkostno razlikujejo v sintaksi skriptnega jezika. Nekaj več razlike pa je v knjižnicah gradnikov, ki so pri posameznem jeziku na voljo.

V nadaljevanju bom kot primer prikazala postopek izdelave karte Slovenije. Zaradi preglednosti je programska koda razdeljena na štiri dele, torej na štiri skriptne datoteke:

- **JScript2**, ki vsebuje deklaracije globalnih spremenljivk;
- **JScript1**, ki vsebuje funkcije;
- **JScript01**, ki ustvari relacijsko bazo in jo pripravi;
- **JScript3**, ki vsebuje ustrezne postopke za analizo ter obdelavo geografskih podatkov.

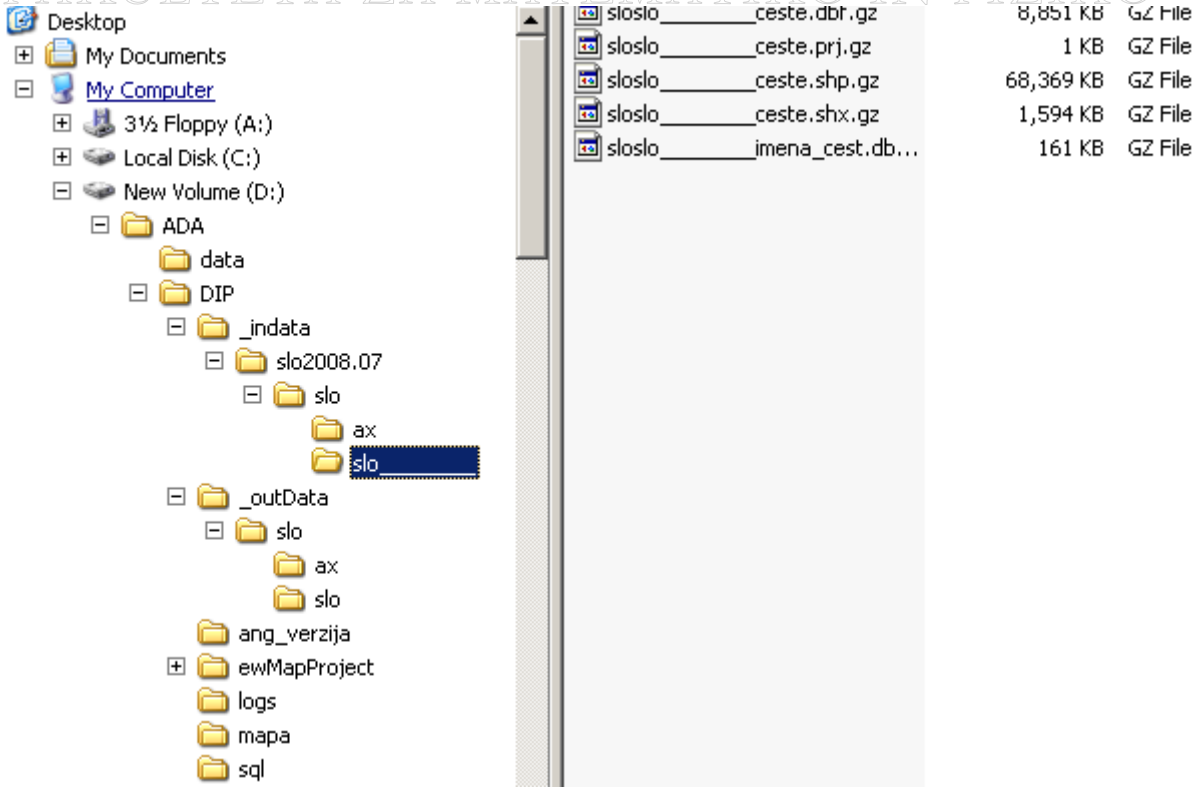
V imeniku *D:\ADA\DIP_indata\slo2008.09* (glej Slika 25 in Slika 26) imamo shranjene dane geografske podatke za Slovenijo. Ti so shranjeni tako, da imamo v imeniku

- *D:\ADA\DIP_indata\slo2008.07\slo* geografske podatke o obliki Slovenije;
- *D:\ADA\DIP_indata\slo2008.07\slo\slo_____* geografske podatke o slovenskih cestah in njihovih oznakah.

Datoteke, v katerih so shranjeni ti geografskih podatki, so oblike DBF, SHP in SHX. Torej, kot že vemo:

- **SHP** (ang. *shape*) predstavlja obliko geografskega objekta;
- **DBF** (ang. *data base file*) vsebuje bazo podatkov;
- **SHX** (ang. *shape index*) indeksna datoteka za povezovanje SHP in DBF datotek.

Slika 25 Imenik vhodnih geografskih podatkov za Slovenijo



Slika 26 Imenik vhodnih geografskih podatkov slovenskih cest in njihovih oznakah

Ker je potrebno dane podatke ustrezno obdelati, bomo obdelane podatke shranili v nov imenik *D:\ADA\DIP_outData\slo* tako, da bodo v podimeniku

- *ax* shranjeni obdelani geografski podatki o obliki Slovenije;
- *slo* shranjeni obdelani geografski podatki o slovenskih cestah in njihovih oznakah.

Več o obdelavi in analizi geografskih podatkov za Slovenijo bo v nadaljevanju prikazano na primerih. Programi so pripravljeni tako, da bi jih lahko brez večjih sprememb uporabili tudi za izdelave kart večih držav. Zato bodo določeni postopki morda videti neoptimalni (uporaba zanke po državah, mi pa bomo imeli le eno državo ... in podobno).

4.2 JSCRIPT2.JS

Datoteka **JScript2** vsebuje deklaracije globalnih spremenljivk, ki jih bomo uporabili v ostalih skriptnih datotekah.

```
/*=======================================================
Globalne spremenljivke
=======================================================
*/

//Imenik kjer so shranjeni izvorni podatki
var imenikVhodPodatkov = "D:\\ADA\\DIP\\_inData";
//Imenik kamor se generirajo podatki
var imenikIzhodnihPodatkov = "D:\\ADA\\DIP\\_outData\\";
//Območje podatkov (EUR-Evropa,USA-Amerika,AFR-Afrika,ASI-Azija,…)
var obmocjePodatkov = "EUR";
//Verzija podatkov
var verzijaPodatkov = "2008.07";

//PostgreSQL bin imenik
```

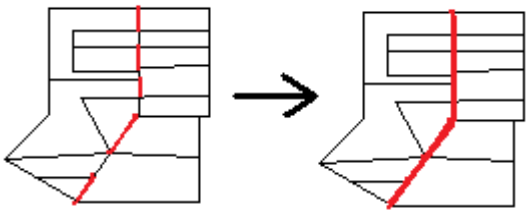
```
var psqlLokacija = "c:\\PROGRA~1\\PostgreSQL\\8.3\\bin\\";
//PostgreSQL port
var psqlPort = "5433";
//Ime relacijske baze, ki jo upravljamo s sistemom PostgreSQL
var psqlGeoDatabase = "_slovenija";
//Ime relacijske baze, ki jo upravljamo s sistemom PostgreSQL
var psqlDatabase = "_slovenija";
//PostgreSQL tablespace
var psqlTableSpace = "ddiskspace";

//Podatki o Lamb.proj. imajo v tabeli spatial_ref_s SRID = 4326
var lambretProj = 4326;
//Podatki o Merc.proj. imajo v tabeli spatial_ref_s SRID = 54004
var mercatorProj = 54004;

var privzetAxDrzave = ['a0'];

/*
Tabela držav
Primer
slo: {
        ime: 'slo',                    //Ime države
        kodnaTabela: 'ISO-8859-2',     //Vrsta kodiranja vhodnih podatkov
        dist: ['slo'],                 //Seznam distribucij znotraj države
    }
*/
var tabDrzava = {
    slo: {
        ime: 'slo',
        kodnaTabela: 'ISO-8859-2',
        dist: ['slo'],
        privzetZnak: 'SRBW',
        znak: [
            [2,1,'LRWG'],[3,3,'LRWB'],[4,4,'MRBY']
        ]
    }
}
```

V obliki dvojne tabele (to je tabela tabel `tabDrzava`) smo definirali imena držav. V našem primeru je to samo Slovenija. Kot vidimo, je v tabeli `tabDrzava` definiran le element `slo`. V tej tabeli `slo` pa imamo definirane podatke o

- oznaki države Slovenije, za nas ima `ime` vrednost `slo`;
- vrsti uporabljenega kodiranja na datoteki;
- seznamu distribucij `dist` znotraj države, v našem primeru je to samo `slo`. Nekatere države so zaradi njihove velike površine razdeljene na bloke (distribucije). Npr. Italija je razdeljena na dve distribuciji (`it1` in `it2`);
- obliki splošne cestne oznake za Slovenijo: `SRBW` – *Small Rectangle Blue and White*;
- obliki vseh dodatnih uporabljenih cestnih oznak- V Sloveniji so to:
  - `LRWG` – *Large Rectangle White and Green* (npr. avtocesta z zeleno oznako in belim napisom A1);
  - `LRWB` – *Large Rectangle White and Blue* (npr. hitra cesta z modro oznako in belim napisom H3);
  - `MRBY` – *Medium Rectangle Blue and Yellow* (npr. regionalna cesta z belo oznako in rumenim napisom 106).

4.3 JSCRIPT1.JS

Datoteka **JScript1.js** vsebuje funkcije, ki omogočajo izpis in obdelavo geografskih podatkov. Pri izvajanju programske kode, ki je zapisana v datoteki **JScript1.js**, se bodo sporočila shranjevala v tekstovno datoteko, kot tudi zapisala v konzolno okno. Funkciji, ki omogočata shranjevanje, sta `ustvariLogDatoteko` in `izpis`. Funkcija `ustvariLogDatoteko` ustvari prazno tekstovno datoteko v imeniku `logs`, `izpis` pa v to datoteko in v konzolno okno zapiše sporočila. Datoteka **JScript1.js** vsebuje tudi funkcije za obdelavo geografskih podatkov, kot so funkcije za uvoz in izvoz geografskih podatkov, funkcije, ki ustvarijo ustrezne indekse v relacijski bazi, ter tudi funkcijo, ki poskrbi za združevanje linij. Ta funkcija poskrbi, da so vsi geometrijski objekti, ki predstavljajo dano ulico, združeni v en geometrijski objekt, tako kot prikazuje Slika 27.



Slika 27 Primer združevanja linij

Namen vseh teh omenjenih funkcij je, da geografske podatke, ki so shranjeni v **shape datotekah** shranimo v relacijsko bazo, kjer jih lahko ustrezno obdelamo, nato obdelane podatke shranimo v nove **shape datoteke**. Pri izvajanju programske kode bomo sporočila shranjevali v tekstovno datoteko in tudi zapisali v konzolno okno. Sporočila bomo shranjevali v tekstovno datoteko zato, da bomo lahko kadarkoli pregledali, kaj se je dogajalo pri izvajanju programske kode. Uporabili bomo tudi funkcijo, ki združi linije, to pa zato, da bomo v tabeli zmanjšali število geometrijskih podatkov tipa **LineString** in s tem bomo dosegli pohitritev sistema.

Poglejmo si vsebino programske kode v datoteki **JScript1.js.**

```
var _shell = WScript.CreateObject("Wscript.Shell");
var _out =  WScript.StdOut;
```

Funkcija `ustvariLogDatoteko` v imeniku `logs` ustvari prazno tekstovno datoteko. Ime datoteke bo oblike `2009_1_20_10_23_ime.log`. Ustrezni datum in čas se generirata avtomatsko, glede na trenutek, ko se funkcija izvede, ime pa je poljubni niz, ki ga podamo kot vhodni podatek. Funkcija vrne referenco na to ustvarjeno datoteko.

```
function ustvariLogDatoteko(datZopisPodatki)
{
    var trenutniCas = new Date();
    var imeLogDatoteke = trenutniCas.getFullYear() + "_" +
    (trenutniCas.getMonth()+1) + "_" + trenutniCas.getDate() + "_"
    + trenutniCas.getHours() + "_" + trenutniCas.getMinutes() + "_" +
    datZopisPodatki + ".log";
    var logDatoteka = fso.CreateTextFile(".\\logs\\" + imeLogDatoteke,
    true);
    return logDatoteka;
}
```

Kot smo že povedali, je namen te funkcije, da v izbranem imeniku ustvari tekstovno datoteko, kamor bomo zapisovali sporočila, ki jih dobimo ob izvedbi programske kode. Sporočila, ki jih dobimo v konzolnem oknu, lahko vidimo le v tistem času, ko se izvaja programska koda. Vendar želimo, da so

ta sporočila shranjena, da jih lahko pregledamo kadarkoli. Zapis sporočil v tekstovno datoteko in v konzolno okno nam bo omogočila funkcija `izpis`.

Funkcija `izpis` zapiše sporočila v tekstovno datoteko in v konzolno okno. Podamo ji niz, ki vsebuje sporočilo, ki ga želimo izpisati.

```
var _printIdent = 0;
function izpis(sporocilo)
{
    var napaka = "";
    for(var i=0; i<_printIdent; ++i)
        napaka += "   ";
    _logDatoteka.WriteLine(napaka + sporocilo + " ");
    _out.WriteLine(napaka + sporocilo + " ");
}
```

V funkciji smo uporabili globalni spremenljivki `_logDatoteka` in `_out`. Slednja je povezana s konzolnim oknom. Prva spremenljivka določa datoteko, kamor se izpisujejo sporočila. Z dejansko datoteko jo v **JScript01.js** in **JScript3.js** povežemo s pomočjo funkcije `ustvariLogDatoteko`. Da so sporočila ustrezno zamaknjena, skrbi globalna spremenljivka `_printIdent`.

Funkcija `zacetekAkcje` zapiše sporočila v tekstovno datoteko in v konzolno okno. Podamo ji niz, ki vsebuje sporočilo, ki ga želimo izpisati.

```
var _startMessage = [];
function zacetekAkcije(sporocilo, casIzvajanja)
{
    var tabpodatkov = {text: sporocilo, time: new Date()};
    var niz = "";
    if(casIzvajanja)
        niz = "ZACETEK " + tabpodatkov.text + " OB " + tabpodatkov.time;
    else
        niz = tabpodatkov.text;
    izpis(niz);
    _startMessage.push(tabpodatkov);
    _printIdent++;
}
```

V funkciji smo uporabili objekt razreda **Array**, ki predstavlja tabelo vrednosti, ki so lahko različnih tipov. V našem primeru sta to sporočilo in nov objekt razreda **Date**, ki predstavlja ta trenutek. Število, ki ga hrani objekt razreda **Date**, je število milisekund, ki so minila od 1. januarja 1970 ob 0:00 UTC. Pri tem je UTC (*Universal Time Coordinated*) oznaka časovne cone, ki je bolj znana kot GMT (*Greenwich Mean Time*). 1. januarja 1970 ob 0:00 UTC ustreza število 0, negativna števila predstavljajo trenutke pred tem časom, pozitivna pa po tem času.

Če želimo slediti času izvajanja (spremenljivka `casIzvajanja`), se v sporočilo poleg ustreznega besedila izpiše še čas začetka akcije.

Z metodo **push** iz razreda **Array** tabelo, ki jo uporabimo za parameter, dodamo na konec tabele `_startMessage`. Da so sporočila, ki se izpišejo v sklopu te akcije ustrezno zamaknjena, pa povečamo globalno spremenljivko `_printIdent`.

Funkcija `konecAkcije` zapiše sporočila o času, ki je potekel, v tekstovno datoteko in v konzolno okno.

```
function konecAkcije(casIzvajanja)
{
    _printIdent--;
    var tabpodatkov = _startMessage.pop();

    if(!casIzvajanja)
        return;
    var trenutniCas = new Date();
    var skupencas = trenutniCas.getTime() - tabpodatkov.time.getTime();
    skupencas = skupencas / 1000.0;
    var niz = "KONEC " + tabpodatkov.text + " OB " + trenutniCas +
    " CAS TRAJANA " + skupencas + " SEKUND";
    izpis(niz);
}
```

Ker smo s tem nehali izpisovati sporočila v sklopu te akcije in se moramo vrniti na prejšni nivo, pomanjšamo globalno spremenljivko `_printIdent`.

Nato z metodo **pop** razreda **Array** iz tabele `startMessage` odstranimo in vrnemo njen zadnji element. Ker smo v ta element shranili čas začetka akcije, lahko sedaj izračunamo čas izvajanja te akcije. Funkcija `konecAkcije` nam tako izpiše sporočilo, ki ga dobi iz tabele `tabpodatkov` in izračunani čas izvajanja.

Namen funkcij `zacetekAkcije` in `konecAkcije` je, da bomo po končanem izvajanju programske kode vedeli, kdaj sta se funkciji začeli in končali izvajati, ter koliko časa je preteklo od klica funkcije `zacetekAkcije` do klica `konecAkcije`. Ti funkciji bomo v nadaljevanju uporabljali predvsem zato, da bomo vedeli, koliko časa je trajal nek postopek pri delu z geografskimi podatki. S tem bomo dobili informacijo, kje je potrebno (če je to seveda možno) pohitriti delo z geografskimi podatki.

Funkcija `zagonUkaza` zažene ukaz v konzolnem oknu in zapiše sporočilo o tem v datoteko, kjer se beležijo izvedene akcije in v konzolno okno. Funkcija nam po končanem izvajanju ukaza zapiše tudi čas trajanja posameznega ukaza. Funkciji kot parameter podamo niz, ki vsebuje ukaz, ki ga želimo zagnati v konzolnem oknu.

```
function zagonUkaza(ukaz)
{
    var cas = new Date();
    izpis("------ zacetek izvajanja");
    izpis("      " + ukaz);

    var o = _shell.Exec("%ComSpec% /c \"" + ukaz + "\"");
    while(!o.StdOut.AtEndOfStream)
        izpis(o.StdOut.ReadLine());

    izpis("      " + ((new Date()).getTime() - cas.getTime()) / 1000.0
    + "s");
    izpis("---------------------- ");
}
```

Zagon ukaza `ukaz` v ukazni vrstici nam omogoča metoda **Exec**, ki izvede aplikacijo v konzolnem oknu in s tem omogoči dostop do objekta **StdOut**. V kodi nam **%ComSpec%** omogoča, da iz programa preusmerimo ukaze za izvedbo v konzolno okno, kjer se izvedejo. Tako se v našem primeru

v konzolnem oknu izvede ukaz `ukaz` in se izpiše vsa informacija tega ukaza. Ko se `ukaz` v konzolnem oknu izvede, nam funkcija v konzolno okno in v tekstovno datoteko izpiše čas trajanja izvedbe ukaza `ukaz`.

Za vsako izvršitev ukaza v ukazni vrstici nam funkcija `zagonUkaza` zapiše čas delovanja posameznih ukazov. To je dobro predvsem zato, ker tako vemo, kje je potrebnoizboljšati programsko kodo …

Funkcija `shpDbfShx` vrne tabelo imen datotek oblike ime.SHP, ime.DBF in ime.SHX – skratka, vrne ime **shape datotek**. Funkcijo bomo uporabljali v funkcijah `priskrbiDatoteke` in `pobriseDatoteke`, v katerih je potrebno določiti imena datotek, ki vsebujejo geografske podatke.

```
function shpDbfShx(ime)
{
    return [ime + '.shp', ime + ".dbf", ime + ".shx"];
}
```

Funkcija `priskrbiDatoteke` kopira datoteke v nov imenik in jih tam razpakira. Podamo ji ime imenika, kjer so datoteke, ime imenika ter kam želimo kopirati datoteke in tabelo imen datotek, ki jih želimo prepisati. Tabelo imen datotek dobimo tako, da v parametru `datoteke` uporabimo klic funkcije `shpDbfShx`, ki nam vrne tabelo imen **shape datotek**, ki so shranjene v imeniku, katerega kot niz podamo v parametru `lokacijaVhodP`.

```
function priskrbiDatoteke(lokacijaVhodP, lokacijaIzhodP, datoteke)
{
    try
    {
        zagonUkaza("mkdir " + lokacijaVhodP);
        for(var i=0; i<datoteke.length; ++i)
        {
            if(typeof datoteke[i] != 'string')
            {
                priskrbiDatoteke(lokacijaVhodP, lokacijaIzhodP,
                 datoteke[i]);
                continue;
            }
            if(fso.FileExists(lokacijaIzhodP + "\\" + datoteke[i]))
            {
                continue;
            }
            try
            {
                zagonUkaza("copy /Y " + lokacijaVhodP + "\\*" + datoteke[i]
                 + ".gz " + lokacijaIzhodP + "\\");
                zagonUkaza("gzip -d " + lokacijaIzhodP + "\\*" +
                 datoteke[i] + ".gz");
            }
            catch(ex)
            {
                izpis("EXCEPTION " + ex.message + " " + datoteke[i]);
            }
        }
        var f = fso.GetFolder(lokacijaIzhodP);
        var fc = new Enumerator(f.files);
        for (; !fc.atEnd(); fc.moveNext())
        {
            fn = new String(fc.item().name);
            if (fn.indexOf("_______") > -1)
```

```
            {
                fc.item().name = fn.substring(14, fn.length);
            }
        }
    }
    catch(ex)
    {
        izpis("EXCEPTION " + ex.message + " " + datoteke);
    }
}
```

Z izvedbo ukaza **mkdir** v ukazni vrstici ustvarimo novo podmapo na trenutnem nahajališču `lokacijaVhodP`. Nato moramo za vsa imena v parametru datoteke preveriti, ali gre za imena "navadnih" datotek, ali pa spet za nek spisek datotek. To ugotovimo tako, da uporabimo operator **typeOf**. Če gre spet za spisek (tabelo) in ustrezni element tabele ni tipa `string`, zato pokličemo funkcijo rekurzivno. Nato preverimo, ali datoteke, ki jih preverjamo obstajajo v imeniku. To storimo z metodo **FileExists**. C**ontinue** uporabimo znotraj zank zato, da omogočimo predčasno prekinitev trenutne ponovitve zanke in nadaljujemo z naslednjo ponovitvijo (če je pogoj za ponovitev zanke še vedno izpolnjen).

Po preverjanju z ukazom **copy** v ukazni vrstici kopiramo eno ali več datotek na drugo mesto. V našem primeru iz imenika vhodnih datotek v imenik, ki smo ga ustvarili z ukazom **mkdir**. Kopirane datoteke je potrebno le še razpakirati. To storimo z ukazom **gzip** v ukazni vrstici. Nato imena datotek v novem imeniku še preimenujemo, ampak le tista, ki imajo v imenu niz _____ tako, da z **substring** ustvarimo nov niz, ki je podniz našega niza (ime datoteke). Podniz določimo prek dveh indeksov.

Funkcija `pobriseDatoteke` iz določenega imenika pobriše datoteke, ki jih ne potrebujemo več. Podamo ji imena datotek kot tabelo nizov in lokacijo imenika, kjer so te datoteke.

Tabelo imen datotek dobimo tako, da v parametru `datoteke` uporabimo klic funkcije `shpDbfShx`, ki nam vrne tabelo imen **shape datotek**, katere so shranjene v imeniku in ga podamo kot niz v parametru `imenik`.

```
function pobriseDatoteke(imenik, datoteke)
{
    for(var i=0; i<datoteke.length; ++i)
    {
        if(typeof datoteke[i] != 'string')
        {
            pobriseDatoteke (imenik, datoteke[i]);
            continue;
        }
        zagonUkaza("del /Q " + imenik + "\\" + datoteke[i]);
    }
}
```

Z ukazom **del /Q** v ukazni vrstici zbrišemo iz imenika eno ali več datotek. Opcija **/Q** nam omogoča, da ukaz ne izpisuje nobenih sporočil o tem, kaj počne.

Za zagon SQL ukaza iz ukazne vrstice v relacijski bazi nam PostgreSQL omogoča uporabo ukaza **psql**, ki se poveže z relacijsko bazo. Ukaz, omogoča uporabo še nekaterih opcij, ki so obvezne:

- **-q** <*quiet*> – omogoči, da se pri izvršitvi ukaza ne izpišejo sporočila;
- **-d** <*dbname*> – ime relacijske baze;
- **-p** <*port*> – vrata za povezavo do relacijske baze;
- **-U** < *username*> – uporabniško ime za povezavo do relacijske baze;
- **-c** <*command*> – parametru sledi SQL ukaz, ki naj se izvede;
- **-f** <*filename*> – pove, da so SQL ukazi v datoteki.

Ta ukaz **psql** uporabimo v dveh funkcijah. Prva omogoča izvedbo posameznega ukaza, druga pa skupine SQL ukazov, zapisanih na datoteki.

Funkcija `izvediSqlUkaz` izvede SQL ukaz v izbrani relacijski bazi. Oba parametra sta niza. Prvi je ime relacijske baze in drugi SQL ukaz, ki se bo izvedel v relacijski bazi. V našem primeru bodo ukazi SQL ukazi, ki ustvarijo relacijsko bazo (`CREATE DATABASE`), počistijo oz. ustvarijo tabelo v relacijski bazi (`DROP TABLE`, `CREATE TABLE`), dodajo oz. odstranijo stolpec v tabeli (`ADD COLUMN`, `DROP COLUMN`), ter ukazi za spreminjanje oz. brisanje geografskih podatkov v tabeli (`UPDATE`, `DELETE`).

```
function izvediSqlUkaz(database, ukaz)
{
    var port = psqlPort;
    zagonUkaza(psqlLokacija + "psql -q -d " + database + " -p " + port +
    " -U postgres -c \"" + ukaz + "\"");
}
```

Funkcija `izvediSqlIzDatoteke` izvede SQL ukaze, ki so zapisani v datoteki. Funkcija dobi kot parametra dva niza – ime relacijske baze in ime datoteke, ki vsebuje SQL ukaze.

```
function izvediSqlIzDatoteke(database, datoteka)
{
    var port = psqlPort;
    zagonUkaza(psqlLokacija + "psql -q -d " + database + " -p " + port +
    "-U postgres -f " + datoteka);
}
```

Funkcija je povsem podobna prejšnji funkciji, le da smo namesto opcije `-c` uporabili opcijo `-f`, ki pove, da so SQL ukazi v datoteki. Ti funkciji bomo uporabljali za poizvedbe po relacijski bazi.

Funkcija `uvoziDbfDatoteke` pretvori **shape datoteke** v SQL datoteke. V teh datotekah so ukazi, ki omogočajo vnos v relacijsko bazo. Podamo ji ime relacijske baze in ime tabele, v katero se bodo shranili podatki ter lokacijo imenika, kjer so **shape datoteke** in oznako uporabljene kodne tabele.

```
function uvoziDbfDatoteke(database, tabela, dbfDatoteka, kodnaTabela)
{
    uvoziShpDatoteke(database, tabela, dbfDatoteka, kodnaTabela, "-n");
}
```

V razdelku 3.1 smo povedali, kako pretvoriti **shape datoteke** v datoteke z SQL ukazi. V primeru, da nimamo skupka vseh treh datotek SHP, DBF in SHX, poimenovanega **shape datoteka**, ampak imamo le datoteko DBF, smo v razdelku 3.1 povedali, da moramo ukaz `shp2pgsql` klicati s parametrom `-n`. Funkcija le pokliče kliče funkcijo `uvoziShpDatoteke`, ki jo opisujemo v nadaljevanju. Pri klicu uporabimo parameter `"-n"`. V našem primeru bomo to funkcijo uporabili le pri uvozu datoteke `imena_cest.dbf`, ki vsebuje podatke o oznakah slovenskih cest in ne vsebuje nobenih drugih datotek, ki sestavljajo **shape datoteko**.

Funkcija `uvoziShpDatoteke` pretvori **shape datoteke** v SQL datoteke. Na teh datotekah so ukazi, ki omogočajo vnos v relacijsko bazo. Funkciji podamo ime relacijske baze, ime tabele, v katero se bodo shranili podatki ter imenik **shape datotek**, iz katerih želimo podatke uvoziti v relacijsko bazo. Podamo tudi vrsto uporabljene kodne tabele in možnost dodati opcije, ki jih ponuja program **shp2pgsql.exe**. Namen te funkcije je uvoz geografskih podatkov, ki so shranjeni v **shape datotekah** v relacijsko bazo, kjer jih lahko ustrezno obdelamo, analiziramo ... Za več informacij o uvozu geografskih podatkov v relacijsko bazo glejte poglavje 3.1.

```
function uvoziShpDatoteke(database, tabela, shpDatoteka, kodnaTabela,
parameter)
{
    var tempDatoteka = "_temp" + tabela + ".sql";
    var proj = " -s " + lambretProj;
    var dbfDatoteka = dbfDatoteka || false;
    var ukaz = psqlLokacija + "shp2pgsql -c -D " + proj + " " + shpDatoteka
               + " " + tabela;
    if(parameter)
        ukaz += " " + parameter;
    if(kodnaTabela)
        ukaz += " -W " + kodnaTabela;
    ukaz += " > " + tempDatoteka;

    zagonUkaza("del " + tempDatoteka);
    zagonUkaza(ukaz);
    izvediSqlIzDatoteke(database, tempDatoteka);
         zagonUkaza("del " + tempDatoteka);
         if(!dbfDatoteka)
         {
             izpis("transforming");
             izvediSqlUkaz(database, "ALTER TABLE " + tabela +
            " DROP CONSTRAINT enforce_srid_the_geom;");
             izvediSqlUkaz(database, "UPDATE " + tabela +
            " SET the_geom = transform(the_geom," + mercatorProj + ");");
         }
}
```

V poglavju 3.1 smo že povedali kako se geografske podatke uvozi v relacijsko bazo. Da si osvežimo spomin, povejmo še enkrat. Ukaz, ki se izvede v ukazni vrstici, je shranjen v spremenljivki `ukaz`. Ker so vsi podatki shranjeni v Lambertovi projekciji, se v ukazu `ukaz` vedno uporabi parameter `-s` z vrednostjo, ki je shranjena v spremenljivki `lambretProj`, ta pa ima vrednost `4326`. V primeru, da geografski podatki niso shranjeni v obliki kodiranja UTF-8, uporabimo parameter `kodnaTabela`, ki v `ukaz` doda parameter `-W` in vrednost kodne tabele. V primeru, da želimo v ukazu `ukaz` uporabiti še dodatne opcije, ki jih ponuja program **shp2pgsql.exe**, uporabimo parameter `parameter`, ki v `ukaz` doda izbrani parameter.

Najprej torej sestavimo ustrezen ukaz. Nato z ukazom **del**, ki se izvede v ukazni vrstici, pobrišemo na trenutnem nahajališču datoteko, katere ime je shranjeno v spremenljivki `tempDatoteka`. To storimo zato, ker se pri izvedbi ukaza `ukaz` ustvari datoteka ravno s tem imenom. Ker datoteka `tempDatoteka` pred izvedbo ukaza `ukaz` lahko že obstaja, jo zato prej izbrišemo.

Nato s klicem funkcije `zagonUkaza` izvedemo ustrezno obliko programa `shp2pgsql`. Ta ustvari datoteko `tempDatoteka`, v kateri so ukazi, ki omogočajo vnos v relacijsko bazo. Te ukaze izvedemo s funkcijo `izvediSqlIzDatoteke`. Sedaj imamo podatke iz datoteke `tempDatoteka` v relacijski bazi in ker te datoteke ne potrebujemo več, jo izbrišemo. S tem smo geografske podatke iz **shape datoteke** shranili v relacijsko bazo, kjer jih je potrebno še transformirati. O transformaciji geografskih podatkov smo govorili v 2. poglavju.

Funkcija `izvozShpDatoteke` shrani geografske podatke iz relacijske baze v **shape datoteko**. Podamo ji ime relacijske baze, imenik **shape datotek** kamor jih želimo shraniti, ime tabele, ki je shranjena v relacijski bazi ter vrsto geometrijskega tipa in SQL stavek, s katerim omejimo podatke za izpis. Vrsto geometrijskega tipa je potrebno navesti zato, da imamo v **shape datoteki** le geometrijske podatke ene vrste (točke, linije …). Izbor geografskih podatkov, ki jih želimo imeti v novih **shape datotekah,** določimo v parametru `sqlStavek`, ki vsebuje SQL stavek.

```
function izvoziShpDatoteke(database, shpDatoteka, tabela, geomTip,
sqlStavek)
{
    var port = psqlPort;
    sqlStavek = sqlStavek || tabela;

    if(geomTip)
    {
        izvediSqlUkaz(psqlGeoDatabase,
        "SELECT count(*),geometrytype(the_geom) FROM " +  tabela +
        " GROUP BY geometrytype(the_geom)");
        izvediSqlUkaz(psqlGeoDatabase, "DELETE FROM " + tabela +
        " WHERE geometrytype(the_geom) <> '" + geomTip +
        "' AND geometrytype(the_geom) <> 'MULTI" + geomTip + "'");
    }

    zagonUkaza(psqlLokacija + "pgsql2shp -u postgres -f " + shpDatoteka + "
               " + database + " -p " + port + " " + sqlStavek);
}
```

Kadar želimo izvoziti le geometrijske podatke določenega tipa (npr. Point), parameter `geomTip` vsebuje neko vrednost in se torej v pogoju if uvrenoti kot true. Takrat bomo najprej izvedli SQL ukaza. S prvim SQL ukazom dobimo število zapisov, ki pripadajo določenemu geometrijskemu tipu. S tem dobimo le informacijo, koliko vrst geometrijskih tipov in število zapisov posameznih geometrijskih tipov je v tabeli `tabela`. Z drugim ukazom iz tabele `tabela` izbrišemo vse tiste podatke, ki nimajo geometrijskega tipa enakega geometrijskemu tipu `geomTip` oziroma niso zbirke podatkov tega geometrijskega tipa. Ta ukaz naredimo predvsem zato, ker je lahko v tabeli več geometrijskih tipov, mi pa želimo v novi **shape datoteki** le geometrijski tip npr. linija ali zbirko linij. Nato z funkcijo `zagonUkaza` izpišemo te filtrirane podatke v **shape datoteko**.

Funkcija `ustvariGistIndeks` ustvari v izbrani tabeli geometrijski indeks. Podamo ji ime relacijske baze, ime tabele in ime geometrijskega stolpca, na katerem se ustvari GIST indeks.

```
function ustvariGistIndeks(database, tabela, geometrija)
{
    var geometrija = geometrija || "the_geom";
    izvediSqlUkaz(database, "create index " + tabela + geometrija +
    "_gist on " + tabela + " using gist (" + geometrija + ");");
}
```

GIST pomeni *Generalized Search Tree* (posplošeno iskalno drevo) in je osnovna oblika indeksiranja GIS podatkov. GIST se uporabi za pospešitev iskanj na poljubnih podatkovnih strukturah. GIST indeksi indeksirajo stolpce, ki imajo podatke o geometriji geometrijskih objektov, v našem primeru je to stolpec `the_geom` tipa `geometry`. Če želimo ustvariti GIST indeks na stolpcu, ki vsebuje geometrijske podatke (torej podatke tipa LINE, MULTIPOLYGON, POINT in podobne …) moramo izvesti SQL ukaz oblike:

```
CREATE INDEX [ime_indeksa] ON [ime_tabele] USING
GIST([polje_z_geometrijskimi_podatki]);
```

Funkcija `zdruziLinije` v relacijski bazi uporabi funkcijo `realis_dissolve_line`, ki poskrbi, da so vsi geometrijski objekti, ki predstavljajo npr ulico, združeni v en geometrijski objekt tipa **LineSting**. S tem bomo v tabeli zmanjšali število geometrijskih podatkov in dosegli hitrejše poizvedovanje po tabeli.

Funkciji `zdruziLinije` podamo ime relacijske baze, tabele, ki vsebuje geometrijske podatke tipa **LineSting** potrebne za združevanje, ime polja po katerem se bodo združevali geometrijski podatki, SQL ukaz (filter), ki omeji združevanje podatkov v vhodni tabeli ter ime stolpca (drugo) po katerem naj se še združijo geometrijski objekti.

```
function zdruziLinije(database, vhodnaTabela, polje, izhodnaTabela, filter,
drugo)
{
    if(!filter || filter == "")
        filter = "true";
    if(!drugo || drugo == "")
        drugo = polje;
    var ukaz = "select * from realis_dissolve_line('" + vhodnaTabela +
    "','" + polje + "','" + izhodnaTabela + "','" + filter + "','" + drugo
    + "');";
    izvediSqlUkaz(database, ukaz);
}
```

4.4 JSCRIPT01.JS

V datoteki **JScript01** je shranjena programska koda, ki ustvari novo relacijsko bazo in izvrši SQL datoteki, s katerima pripravimo relacijsko bazo, v katero bomo lahko shranjevali geografske podatke o Sloveniji. Relacijsko bazo bomo poimenovali kar `_slovenija`, saj bo vsebovala le geografske podatke o Sloveniji. V SQL datoteki `dodajanje_koord_sis` je shranjen SQL ukaz, ki v tabelo `spatial_ref_sys` doda podatke o transverzalni Mercatorjevi projekciji. O dodajanju podatkov transverzalne Mercatorjeve projekcije v tabelo `spatial_ref_sys` smo govorili na strani 20. V SQL datoteki `realis_foo` je shranjen SQL ukaz, ki v relacijski bazi ustvari funkcijo `realis_dissolve_line`. Koda v datoteki **JScript01**, ki izvrši ustvarjanje in pripravo relacijske baze `_slovenija`, je:

```
var fso = WScript.CreateObject("Scripting.FileSystemObject");
f = fso.OpenTextFile("JScript1.js", 1);
var code = f.readAll();
eval(code);
f = null;

/*-------------------------------------------------------------------------
PRIPRAVA BAZE
 Argumenti
      0   datoteka z spremeljivkami             npr "JScript2.js"

 Klic: D:\ADA\DIP>cscript JScript01.js JScript2.js -- ustvarimo bazi
 psqlGeoDatabase in psqlDatabase nato v bazah pripravimo tabele
-------------------------------------------------------------------------
*/

var f = fso.OpenTextFile(WScript.Arguments(0), 1);
var code = f.readAll();
eval(code);
f.Close();
f = null;

var _logDatoteka = ustvariLogDatoteko(psqlGeoDatabase);
```

S pomočjo programa **psql.exe** ustvarimo relacijsko bazo `_slovenija`.

```
zagonUkaza(psqlLokacija + "psql -q -p " + psqlPort + " -U postgres -c \"" +
"CREATE DATABASE " + psqlGeoDatabase
+ " ENCODING='UTF8' TEMPLATE=template_postgis TABLESPACE=" + psqlTableSpace
+ "\"");
```

Ker je potrebno relacijsko bazo `_slovenija` še ustrezno pripraviti za delo z geografskimi podatki, jo pripravimo tako, da izvedemo ukaze, ki so na ustreznih datotekah. Prva `dodajanje_koord_sis.sql` vsebuje SQL ukaz, ki v tabelo `spatial_ref_sys` doda podatke o transverzalni Mercatorjevi projekciji in ima `SRID` vrednost `54004` in je identifikacijska številka te koordinatne projekcije. Druga `realis_foo.sql` vsebuje ukaz, ki v relacijski bazi ustvari funkcijo `realis_dissolve_line`.

```
izvediSqlIzDatoteke(psqlGeoDatabase, "sql\\dodajanje_koord_sis.sql");
izvediSqlIzDatoteke(psqlGeoDatabase, "sql\\realis_foo.sql");
```

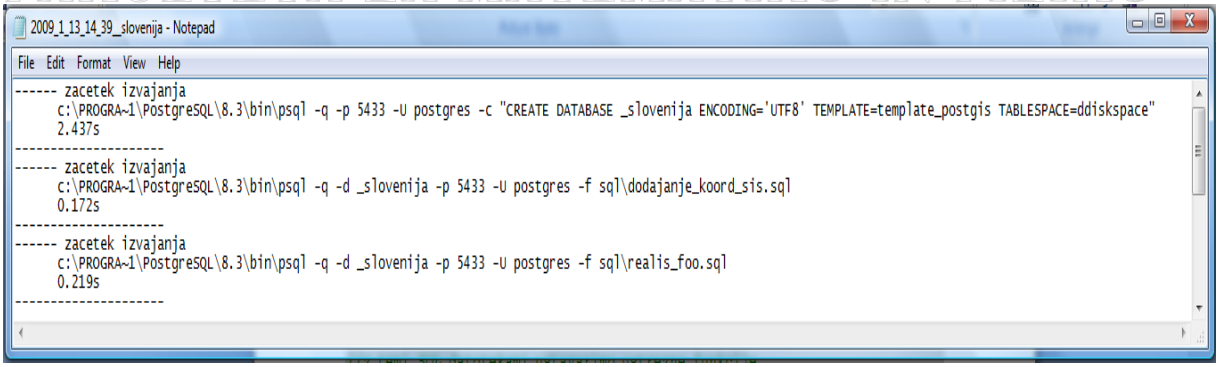
Poglejmo si, kako v konzolnem oknu zaženemo našo programsko kodo, s katero ustvarimo in pripravimo relacijsko bazo podatkov `_slovenija`. To storimo tako, da v ukazni vrstici napišemo ukaz `scsript JScript01.js JScript2.js`, kot prikazuje Slika 28.

Slika 28 Primeri ukazov v ukazni vrstici, ki ustvarijo in pripravijo relacijsko bazo podatkov

Vidimo, da se je ustvarila relacijska baza `_slovenija`. V tabeli `spatial_ref_sys`, ki je v relacijski bazi, so se dodali podatki o transverzalni Mercatorjevi projekciji in v tej bazi se je ustvarila tudi funkcija `realis_dissolve_line`.

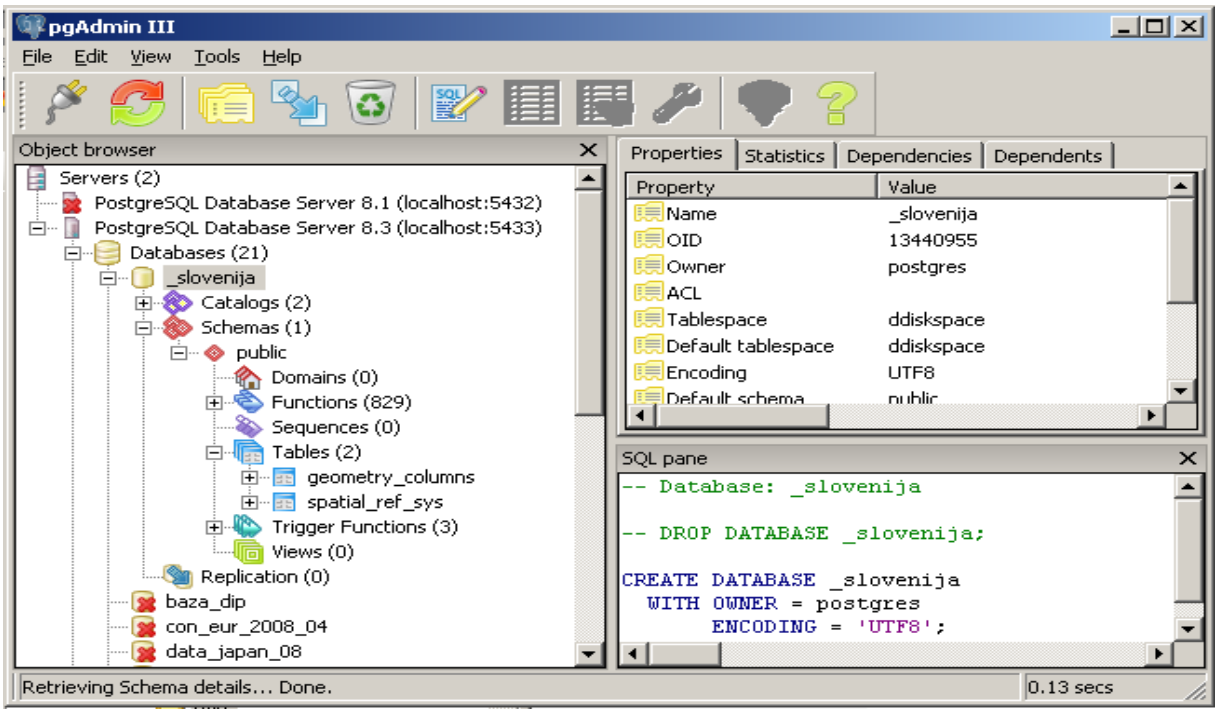
V funkciji `izpis` smo definirali, da se bodo sporočila izpisovala v ukazni vrstici in tudi v tekstovni datoteki. Zato smo pri zagonu programske kode dobili tekstovno datoteko `2009_1_13_14_39_slovenija`. Pri vsakem zagonu programske kode se bo v imeniku `logs` ustvarila nova tekstovna datoteka, ki bo vsebovala sporočila programske kode (Slika 29). Namen te datoteke je, da bomo imeli shranjena sporočila vsake izvedene programske kode, zato da lahko večkrat pogledamo kje so se zgodile napake, problemi … Ker je izpis sporočil v ukazni vrstici popolnoma enak temu v tekstovni datoteki, bom v nadaljevanju prikazala vsak izpis sporočil le z ukazno vrstico.

V našem primeru so izvršitve ukazov hitre in vidimo tudi, da je samo ustvarjanje relacijske baze trajalo le 2,437 sekunde.



Slika 29 Prikaz ukazov v tekstovni datoteki, ki so bili izvršenih v ukazni vrstici

Na sliki Slika 30 si še poglejmo, kako je v PostgreSQL-u videti naša relacijska baza `_slovenija`.



Slika 30 Prikaz relacijske baze _slovenija v PostgreSQL

4.5 JSCRIPT3.JS

Sedaj imamo ustvarjeno in pripravljeno relacijsko bazo `_slovenija`. V relacijsko bazo `_slovenija` bi radi shranili dane geografske podatke za Slovenijo, ki jih imamo pripravljene za obdelavo in nato jih pretvorimo v novo **shape datoteko**. Najprej si poglejmo, kako obdelamo dane geografske podatke za Slovenijo, ki se nahajajo v imeniku *D:\ADA\DIP_indata\slo2008.07\slo\ax*.

Vemo, da so geografski podatki v imeniku stisnjeni v format **GZ** in jih je potrebno razpakirati. To storimo s funkcijo `priskrbiDatoteke`, ki dane geografske podatke kopira v nov imenik *D:\ADA\DIP_outData\slo\ax* in razpakira. Nato razpakirane geografske podatke shranimo v relacijsko bazo `_slovenija`, jih tam obdelamo in na koncu jih zapišemo nazaj v **shape datoteko**. Poglejmo si programsko kodo, ki izvrši obdelavo danih podatkov za Slovenijo.

```
var fso = WScript.CreateObject("Scripting.FileSystemObject");
f = fso.OpenTextFile("JScript1.js", 1);
```

```
var code = f.readAll();
eval(code);
f.Close();

var ewm = WScript.CreateObject("ewMapCom.LibGlobal");

/*
---------------------------------------------------------------------------
Argumenti
      0   datoteka s spremenljivkami   npr "JScript2.js"
      1   akcija                       npr " obdelava_drzave " ali "
                                       obdelava_drzave, obdelava_cest,…"
                                       ali vse" (glej spremenljivko akcije)
      2   država                       npr "slo" ali "slo,hrv,…" ali "vse"

Klic: cscript JScript3.js JScript2.js obdelava_drzave slo
---------------------------------------------------------------------------
*/

//Odpremo datoteko z spremenljivkami
var f = fso.OpenTextFile(WScript.Arguments(0), 1);
var code = f.readAll();
eval(code);
f.Close();
f = null;

var _logDatoteka = ustvariLogDatoteko(psqlGeoDatabase);
```

Z akcijo `obdelava_drzave` shranimo geografske podatke o Sloveniji v relacijsko bazo `_slovenija`. Z akcijo `obdelava_cest` v to relacijsko bazo shranimo še geografske podatke o slovenskih cestah. Shranjene geografske podatke bomo v relacijski bazi ustrezno obdelali in nato obdelane geografske podatke shranili v novo **shape datoteko**.

```
var akcije = ['obdelava_drzave','obdelava_cest'];

if(WScript.Arguments(1) != 'all')
    akcije = WScript.Arguments(1).split(',');

//ARGUMENT 2 - država
var drzave = WScript.Arguments(2);
```

V primeru, če je v ukazni vrstici drugi argument `all`, potem v tabelo `drzave` napolnimo s seznamom distribucij `dist` znotraj države. Že prej smo omenili, da je koda napisana splošno, torej tudi za primere, ko imamo znotraj države lahko več distribucij (delov). V našem primeru je distribucija le ena, samo `slo`.

```
if(drzave == 'all')
{
    drzave = [];
    for(var c in dist) //S spremeljivko c se sprehodimo skozi vse lastnosti
        //objekta dist. Pri tem se izvedejo vsi stavki v telesu zanke.
        drzave[drzave.length] = c;
}
else
{
    drzave = drzave.split(",");
}
```

Tu je glavna zanka, ki preteče po vseh državah. V našem primeru je v seznamu držav samo Slovenija.

```
for(var c = 0; c < drzave.length; ++c)
{
    var drzava = drzave[c];
    var imeDistribucije = tabDrzav[drzava].ime;
    var imenikVPodatDrzave = imenikVhodPodatkov + "\\" + imeDistribucije +
         verzijaPodatkov + "\\" + drzava + "\\";
    izpis(tabDrzava[drzava] + " " + drzava);
    var kodnaTab = tabDrzava[drzava].kodnaTabela;

    //Vzamemo vse distribucije v državi
    var distribucije = tabDrzava[drzava].dist;

    //Najprej obdelamo akcije države
    for(var a = 0; a < akcije.length; ++a)
    {
        var akcija = akcije[a];

        //Generalne akcije, ki veljajo za celo državo
        //-------------------------------------------------
```

Z metodo `priskrbiDatoteke` kopiramo datoteke **drzava.*** v imenik D:*\ADA\DIP_outData\slo\ax*. Nato v imeniku pobrišemo datoteke z imenom **_drzava.***. To storimo zato, da kasneje nimamo težav v primeru, da datoteke že obstajajo. Potem v tabelo `_drzava` pretvorimo podatke iz datoteke **drzava.*** in nato tabelo obdelamo z ustreznimi SQL ukazi. Podatke v tabeli `_drzava` obdelamo tako, da iz tabele izbrišemo tiste podatke, ki imajo v stolpcu `namelc` vrednost `UND`, saj teh podatkov ne bomo potrebovali. Potrebujemo le vrstice, ki imajo v stolpcu `namelc` vrednost `SVN` (Slovenija). Na koncu pretvorimo željene podatke iz tabele `_drzava` v shape datoteki **_drzava.***, ki je primerna za geografske podatke. Potem je potrebno še čiščenje nepotrebnih datotek, ki smo jih dobili z metodo `priskrbiDatoteke` .

```
        if(akcija == "obdelava_drzave")
        {
            zacetekAkcije("AKCIJE " + akcija + " DRZAVA " + drzava);

            var axLokacija = imenikIzhodnihPodatkov + drzava + "\\ax";
            var imeDistribucije = tabDrzav[drzava].ime;
            var ax = privzetAxDrzave;

            priskrbiDatoteke (imenikVPodatDrzave + "ax", axLokacija,
            [shpDbfShx('drzava')]);

            zagonUkaza("del " + axLokacija + "\\slo_drzava.*");
            izvediSqlUkaz(psqlGeoDatabase, "DROP TABLE _drzava;");
            uvoziShpDatoteke(psqlGeoDatabase, "_drzava", axLokacija +
             "\\drzava.shp");
            izvediSqlUkaz(psqlGeoDatabase, "DELETE FROM _drzava WHERE
             namelc = 'UND';");
            izvoziShpDatoteke(psqlGeoDatabase, axLokacija +
            "\\slo_drzava.shp", "_drzava", "POLYGON");
            pobriseDatoteke(axLokacija, shpDbfShx('drzava'));

            konecAkcije();
        }
```

Programsko kodo, s katero obdelamo geografske podatke za Slovenijo, zaženemo v ukazni vrstici z ukazom `cscript JScript3.js JScript2.js obdelava_drzave slo`, kot prikazuje Slika 31. Povejmo še, da smo za prvi argument vzeli akcijo `obdelava_drzave` in za drugi argument državo `slo` (Slovenija).

```
D:\ADA\DIP>cscript JScript3.js JScript2.js obdelava_drzave slo
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

[object Object] slo
ZACETEK AKCIJE obdelava_drzave DRZAVA slo OB Wed Jan 14 12:49:14 UTC+0100 2009
   ------ zacetek izvajanja
          mkdir D:\ADA\DIP\_inData\slo2008.07\slo\ax
          0.078s
   ----------------------
   ------ zacetek izvajanja
          mkdir D:\ADA\DIP\_inData\slo2008.07\slo\ax
          0s
   ----------------------
   ------ zacetek izvajanja
          copy /Y D:\ADA\DIP\_inData\slo2008.07\slo\ax\*drzava.shp.gz D:\ADA\DIP\_outData\slo\ax\
   D:\ADA\DIP\_inData\slo2008.07\slo\ax\slo__________drzava.shp.gz
            1 file(s) copied.
          0.203s
   ----------------------
   ------ zacetek izvajanja
          gzip -d D:\ADA\DIP\_outData\slo\ax\*drzava.shp.gz
          0.187s
   ----------------------
   ------ zacetek izvajanja
          copy /Y D:\ADA\DIP\_inData\slo2008.07\slo\ax\*drzava.dbf.gz D:\ADA\DIP\_outData\slo\ax\
   D:\ADA\DIP\_inData\slo2008.07\slo\ax\slo__________drzava.dbf.gz
            1 file(s) copied.
          0.063s
   ----------------------
   ------ zacetek izvajanja
          gzip -d D:\ADA\DIP\_outData\slo\ax\*drzava.dbf.gz
          0.047s
   ----------------------
   ------ zacetek izvajanja
          copy /Y D:\ADA\DIP\_inData\slo2008.07\slo\ax\*drzava.shx.gz D:\ADA\DIP\_outData\slo\ax\
   D:\ADA\DIP\_inData\slo2008.07\slo\ax\slo__________drzava.shx.gz
            1 file(s) copied.
          0.078s
   ----------------------
   ------ zacetek izvajanja
          gzip -d D:\ADA\DIP\_outData\slo\ax\*drzava.shx.gz
          0s
   ----------------------
   ------ zacetek izvajanja
          del D:\ADA\DIP\_outData\slo\ax\slo_drzava.*
          0.046s
   ----------------------
   ------ zacetek izvajanja
          c:\PROGRA~1\PostgreSQL\8.3\bin\psql -q -d _slovenija -p 5433 -U postgres -c "DROP TABLE _drzava;"
          0.829s
   ----------------------
   ------ zacetek izvajanja
          del _temp_drzava.sql
          0.031s
   ----------------------
   ------ zacetek izvajanja
          c:\PROGRA~1\PostgreSQL\8.3\bin\shp2pgsql -c -D  -s 4326 D:\ADA\DIP\_outData\slo\ax\drzava.shp _drzava > _temp_drzava.sql
          0.219s
   ----------------------
   ------ zacetek izvajanja
          c:\PROGRA~1\PostgreSQL\8.3\bin\psql -q -d _slovenija -p 5433 -U postgres -f _temp_drzava.sql
                        addgeometrycolumn
   ------------------------------------------------------------
    public._drzava.the_geom SRID:4326 TYPE:MULTIPOLYGON DIMS:2

   (1 row)

          4.563s
   ----------------------
   ------ zacetek izvajanja
          del _temp_drzava.sql
          0.015s
   ----------------------
   transforming
   ------ zacetek izvajanja
          c:\PROGRA~1\PostgreSQL\8.3\bin\psql -q -d _slovenija -p 5433 -U postgres -c "ALTER TABLE _drzava DROP CONSTRAINT enforce_srid_the_geom;"
          0.266s
   ----------------------
   ------ zacetek izvajanja
          c:\PROGRA~1\PostgreSQL\8.3\bin\psql -q -d _slovenija -p 5433 -U postgres -c "UPDATE _drzava SET the_geom = transform(the_geom,54004);"
          0.672s
   ----------------------
   ------ zacetek izvajanja
          c:\PROGRA~1\PostgreSQL\8.3\bin\psql -q -d _slovenija -p 5433 -U postgres -c "DELETE FROM _drzava WHERE namelc = 'UND';"
          0.172s
   ----------------------
   ------ zacetek izvajanja
          c:\PROGRA~1\PostgreSQL\8.3\bin\psql -q -d _slovenija -p 5433 -U postgres -c "SELECT count(*), geometrytype(the_geom) FROM _drzava GROUP BY geometrytype
(the_geom)"
    count | geometrytype
   -------+--------------
        1 | MULTIPOLYGON
   (1 row)

          0.297s
   ----------------------
   ------ zacetek izvajanja
          c:\PROGRA~1\PostgreSQL\8.3\bin\psql -q -d _slovenija -p 5433 -U postgres -c "DELETE FROM _drzava WHERE geometrytype(the_geom) <> 'POLYGON' AND geometry
type(the_geom) <> 'MULTIPOLYGON'"
          0.171s
   ----------------------
   ------ zacetek izvajanja
          c:\PROGRA~1\PostgreSQL\8.1\bin\pgsql2shp -u postgres -f D:\ADA\DIP\_outData\slo\ax\slo_drzava.shp _slovenija -p 5433 _drzava
   Initializing... Done (postgis major version: 1).
   Output shape: Polygon
   Dumping: XX [1 rows].
          0.704s
   ----------------------
   ------ zacetek izvajanja
          del /Q D:\ADA\DIP\_outData\slo\ax\drzava.shp
          0.015s
   ----------------------
   ------ zacetek izvajanja
          del /Q D:\ADA\DIP\_outData\slo\ax\drzava.dbf
          0.016s
   ----------------------
   ------ zacetek izvajanja
          del /Q D:\ADA\DIP\_outData\slo\ax\drzava.shx
          0.015s
   ----------------------
KONEC AKCIJE obdelava_drzave DRZAVA slo OB Wed Jan 14 12:49:23 UTC+0100 2009 CAS TRAJANA 8.844 SEKUND
```

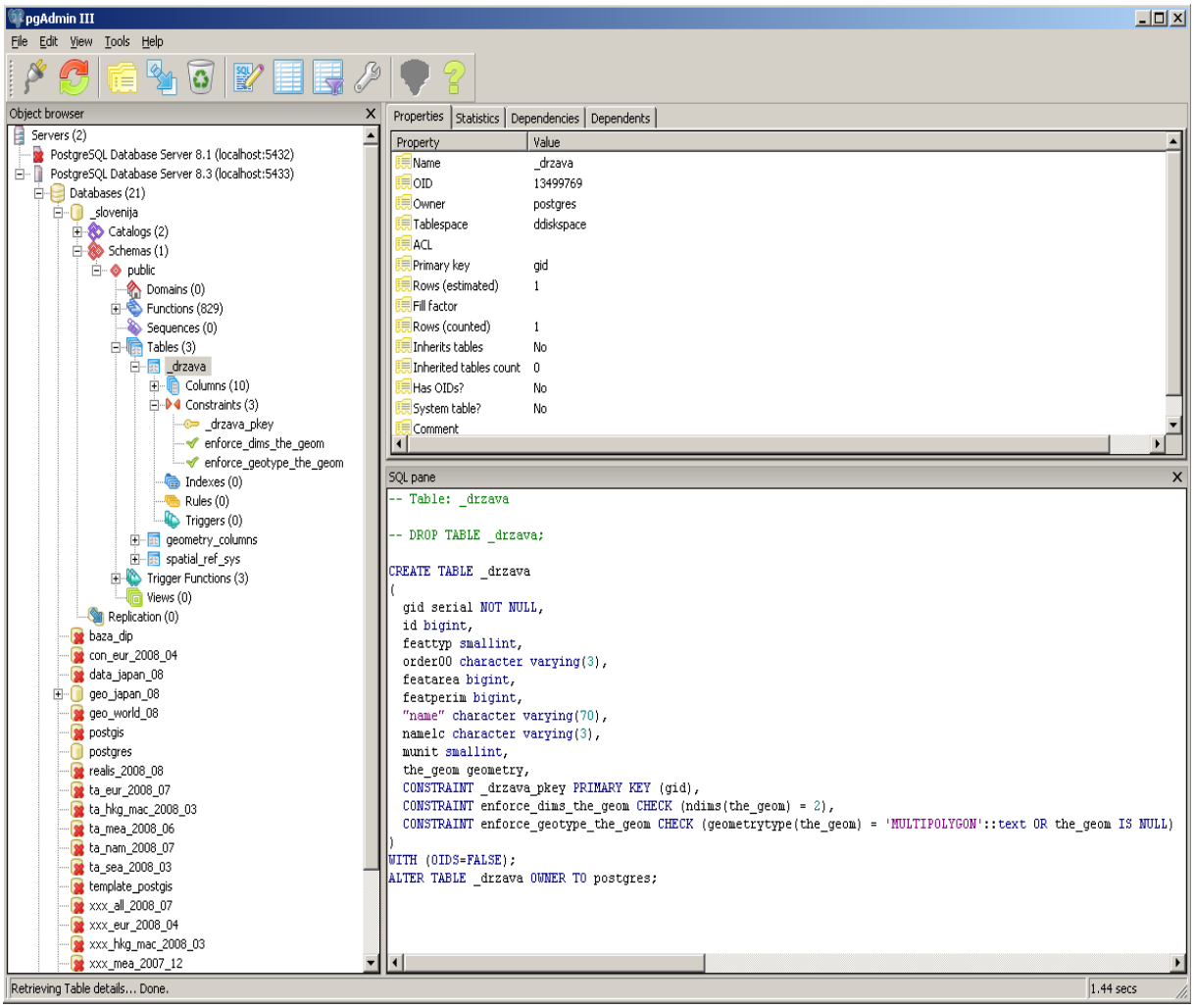
Slika 31 Prikaz izvršene programske kode v ukazni vrstici

Z ukazom v ukazni vrstici smo zagnali programsko kodo, ki je izvršila izbrano akcijo, to je `obdelava_drzave` za Slovenijo.

Z akcijo `obdelava_drzave` smo dobili v novem imeniku **shape datoteko**, ki vsebuje vse podatke o Sloveniji tako, da smo:

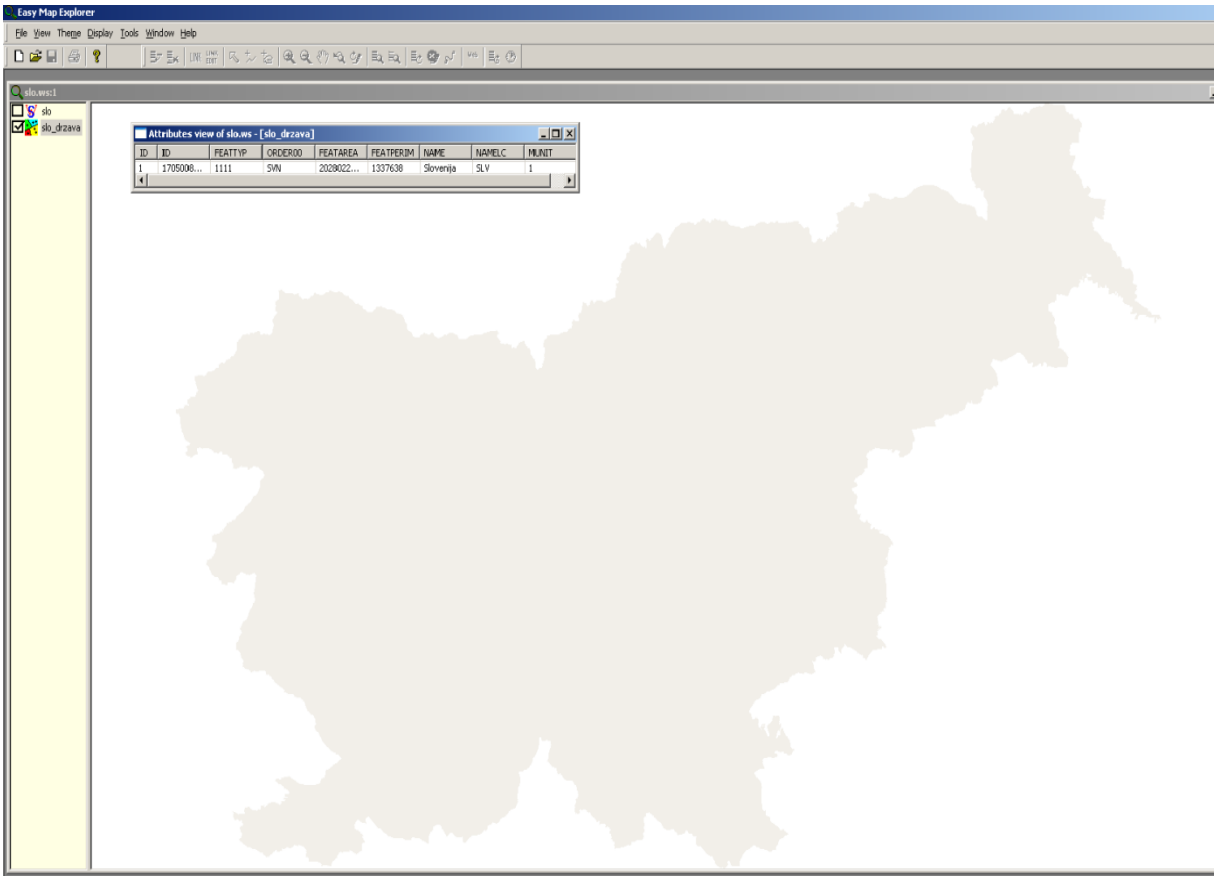
- s funkcijo `priskrbiDatoteke` kopirali dane podatke v nov imenik *D:\ADA\DIP_indata\slo2008.07\slo\ax* in jih tam razpakirali;
- s funkcijo `uvoziShpDatoteke` geografske podatke iz **shape datotek** iz novega imenika prenesli v relacijsko bazo `_slovenija` in naredili transformacijo koordinat iz Lambertove projekcije v transverzalno Mercatorjevo projekcijo;
- v tabeli `_drzava` obdelali podatke z ustreznim SQL ukazom. Skratka, izbrisali smo le tiste podatke, ki imajo v stolpcu `namelc` vrednost `UND`. Več o tem glej na strani 58, ko smo govorili o akciji `obdelava_drzave`;
- s funkcijo `izvoziShpDatoteke` pretvorili obdelane podatke iz tabele `_drzava` v novo **shape datoteko**;
- s funkcijo `pobriseDatoteke` počistili iz novega imenika tiste podatke, ki so se kopirali pri zagonu funkcije `priskrbiDatoteke`.

Poglejmo si rezultat v PostgreSQL-u (Slika 32). Vidimo, da se je v relacijski bazi `_slovenija` ustvarila nova tabela `_drzava`, ki je napolnjena z geografskim podatkom tipa `MultiPolygon`.



Slika 32 Prikaz zgradbe tabele v relacijski bazi _slovenija

Geografski podatek, ki smo ga pridobili z zagonom programske kode, se nahaja v imeniku *D:\ADA\DIP_outData\slo\ax\slo_drzava.**. Poglejmo si vsebino podatkov z GIS orodjem **Easy Map Explorer**. Na Slika 33 vidimo geografski objekt Slovenija (predstavljena kot poligon) in lastnosti Slovenije (prikazani na oknu `Atributes view of - [slo_drzava.shp]`).



Slika 33 Prikaz datotek slo_drzava v GIS orodju Easy Map Explorer

Sedaj smo obdelali le dan geografski podatek za Slovenijo, ki je shranjen v imeniku *D:\ADA\DIP_outData\slo\ax*. Pri tem smo dobili obdelan geografski objekt Slovenija. Poglejmo si še obdelavo geografskih podatkov, ki so shranjeni v imeniku *D:\ADA\DIP_inData\slo2008.07\slo\slo_____*. Programska koda, ki izvrši obdelavo teh geografskih podatkov, je sledeča:

```
//Obdelava dist akcij
for(var d = 0; d < distribucije.length; ++d)
{
    //Postavimo spremenljivke, ki jih rabimo v spodnjih akcijah
    var dist = distribucije[d];
    var lokacija = imenikIzhodnihPodatkov + drzava + "\\" + dist;
    var imeDistribucije = tabDrzava[drzava].ime;
    var distLokacija = imenikVhodPodatkov + "\\" + imeDistribucije +
         verzijaPodatkov + "\\" + dist + "\\" + dist + "_______";
```

V našem primeru imamo le eno distribucijo in to je `slo`, zato bomo šli v zanko le enkrat. V primeru, da bi imeli več distribucij, bi lahko obdelali vsako distribucijo posebej. Ker bomo v nadaljevanju imeli delo z geografskimi podatki o slovenskih cestah in njihovih oznakah, jih bomo zato obdelali s posebno akcijo t.j. `obdelava_cest`.

Z akcijo `obdelava_cest` pripravimo ceste in njihove oznake za Slovenijo tako, da z funkcijo `priskrbiDatoteke` kopiramo dane podatke o cestah in njihovih oznakah, ki so v imeniku `distLokacija` (*D:\ADA\DIP_indata\slo2008.07\slo\slo_____*) in so stisnjeni v format **GZ** v nov imenik `lokacija` (*:\ADA\DIP_outData\slo*) in jih tam razpakiramo.

```
if(akcija == "obdelava_cest")
{
    zacetekAkcije("AKCIJA " + akcija + " DRZAVA " + drzava +
    " DISTRIBUCIJA " + dist);

    priskrbiDatoteke(distLokacija, lokacija,
    [shpDbfShx('ceste'),'imena_cest.dbf']);
    try
    {
```

Akcijo `obdelava_cest` bomo razdelili na več delov in sicer na toliko, kolikor tipov cest poznamo. To storimo predvsem zato, ker želimo geografske podatke o cestah, ki so shranjeni v eni **shape datoteki** (v našem primeru v **_ceste.***) razbiti na več **shape datotek**, ker jih bomo ločili po tipu ceste.

Z akcijo `obdelava_cest-UVOZ` najprej izbrišemo tabelo `_ceste`, v katero se bodo uvozili geografski podatki o slovenskih cestah. To storimo zato, ker tabela `_ceste` lahko že obstaja. Nato z funkcijo `uvoziShpDatoteke` pretvorimo geografske podatke o slovenskih cestah iz novega imenika `lokacija` v relacijsko bazo `_slovenija` in pri tem naredimo transformacijo koordinat geografskih podatkov iz Lambertove projekcije v transverzalno Mercatorjevo projekcijo. O uvozu geografskih podatkov v relacijsko bazo in transformacijo geografskih podatkov smo govorili v poglavjih 2 in 3.1. Nato v tabeli `_ceste` izvedemo še dva SQL ukaza. S prvim izbrišemo geografske podatke, ki nimajo `feattyp` enak `4110` in `4130`, z drugim pa v stolpcu `name` preimenujemo vse, ki imajo izpolnjeno polje v vrednost `null`. S tema ukazoma smo v tabeli `_ceste` počistili tiste podatke, s katerimi se v nadaljevanju ne bomo ukvarjali in za nas niso pomembni in s tem pospešili iskanje podatkov v tabeli.

```
            zacetekAkcije("AKCIJA " + akcija + "-UVOZ" + " DRZAVA " +
            drzava + " DISTRIBUCIJA " + dist);
            izvediSqlUkaz(psqlGeoDatabase, "DROP TABLE _ceste;");
            uvoziShpDatoteke(psqlGeoDatabase, "_ceste", lokacija +
            "\\*ceste.shp", kodnaTab);
            ustvariGistIndeks(psqlGeoDatabase, "_ceste");

            //Odstranimo nekatere ceste
            izvediSqlUkaz(psqlGeoDatabase, "DELETE FROM _ceste WHERE " +
            "feattyp <> 4110 AND feattyp <> 4130;");
            izvediSqlUkaz(psqlGeoDatabase, "UPDATE _ceste SET name = '' +
            " WHERE name <> '' AND nametyp <> 'ON';");
            konecAkcije();
```

Z akcijo `obdelava-TRAJEKTI DRZAVA` najprej zaradi varnosti izbrišemo tabelo `_tmp` v katero se bodo z funkcijo `zdruziLinije` shranili podatki o imenu in geometriji združenih trajektnih linijah. Torej iz tabele `_ceste` bomo združevali trajektne linije po stolpcu `name`, ki vsebuje imena trajektnih linij in združevali bomo le tiste, ki imajo `feattyp=4130`, torej trajekte. Ko imamo združene trajektne linije, potem te podatke z funkcijo `izvoziShpDatoteke` pretvorimo podatke iz tabele `_tmp` v imenik `lokacija` **shape datoteko** `_trajekti.*`. Ker za Slovenijo ni podatkov o trajektnih linijah, se v izpisu izpiše opozorilo:

```
Initializing... ERROR: Cannot determine geometry type (empty table).
```

Tabelo `_ceste` je potrebno še počistiti. To storimo tako, da v tabeli izbrišemo podatke, ki imajo `feattyp=4130`, torej trajekte. V tabeli moramo ustvariti še stolpca, poimenovana `r_type` in `r_itype`. Vsebovala bosta podatke, po katerih bomo posamezne ceste ločili po tipu ceste. Nato pa v tabeli ustvarimo še GIST indeks, s katerim pospešimo iskanje podatkov v tabeli.

```
zacetekAkcije("AKCIJA " + akcija + "-TRAJEKTI DRZAVA " + drzava
+ " DISTRIBUCIJA " + dist);

izvediSqlUkaz(psqlGeoDatabase, "DROP TABLE _tmp;");
zdruziLinije(psqlGeoDatabase, "_ceste", "name", "_tmp",
"feattyp = 4130", null);
izvoziShpDatoteke(psqlGeoDatabase, lokacija + "\\_trajekt.shp",
"_tmp");

izvediSqlUkaz(psqlGeoDatabase,
"DELETE FROM _ceste WHERE feattyp <> 4110;");
izvediSqlUkaz(psqlGeoDatabase,
"ALTER TABLE  ceste DROP COLUMN r type;");
izvediSqlUkaz(psqlGeoDatabase,
"ALTER TABLE _ceste ADD COLUMN r_type int2;");
izvediSqlUkaz(psqlGeoDatabase,
"ALTER TABLE _ceste DROP COLUMN r_itype;");
izvediSqlUkaz(psqlGeoDatabase,
"ALTER TABLE _ceste ADD COLUMN r_itype int2;");
izvediSqlUkaz(psqlGeoDatabase,
"UPDATE _ceste SET r_itype = 0;");
ustvariGistIndeks(psqlGeoDatabase, "_ceste", "r_itype");

konecAkcije();
```

Z akcijo `obdelava_cest-EUROCESTE DRZAVA` najprej zaradi varnosti izbrišemo tabelo `_tmp` v katero se bodo z klicem funkcije `zdruziLinije` shranili podatki o imenu in geometriji združenih evropskih cest. V tabeli `_ceste` najprej izvedemo SQL ukaz s katerim na tabeli `_ceste` nastavimo nekaj podatkov po katerih bomo združevali ceste. Podatke v tabeli spremenimo tako, da v stolpcu `r_type` nastavimo vrednost `0` in v stolpcu `r_itype` nastavimo vrednost `1` le tistim podatkom, ki imajo izpolnjujejo pogoj `rtetyp<3 AND (frc=0 OR (frc=1 AND freeway=1)`. Torej iz tabele `_ceste` bomo ceste združevali po stolpcu `name`, ki vsebuje imena evropskih cest in združevali bomo le tiste, ki imajo `r_type=0`, torej evropske ceste. Ko imamo združene ceste, potem te podatke z funkcijo `izvoziShpDatoteke` pretvorimo podatke iz tabele `_tmp` v imenik `lokacija` **shape datoteko** `_eur_ceste.*`.

```
//Obdelava evropskih avto ceste za državo.
zacetekAkcije("AKCIJA " + akcija + "-EURCESTE DRZAVA " + drzava
+ " DISTRIBUCIJA " + dist);

izvediSqlUkaz(psqlGeoDatabase, "DROP TABLE _tmp;");
izvediSqlUkaz(psqlGeoDatabase, "UPDATE _ceste SET r_type = 0,"
+ "r_itype = 1 WHERE rtetyp < 3 AND (frc = 0 OR (frc = 1 AND" +
" freeway = 1))");
zdruziLinije(psqlGeoDatabase, "_ceste", "name", "_tmp",
"r_type = 0", null);
izvoziShpDatoteke(psqlGeoDatabase, lokacija +
"\\eur_ceste.shp", "_tmp");

konecAkcije();
```

Z akcijo `obdelava_cest-AVTOCESTE DRZAVA` najprej zaradi varnosti izbrišemo tabelo `_tmp` v katero se bodo z klicem funkcije `zdruziLinije` shranili podatki o imenu in geometriji združenih avtocest. V tabeli `_ceste` najprej izvedemo SQL ukaz s katerim na tabeli `_ceste` nastavimo nekaj podatkov. Podatke v tabeli spremenimo tako, da v stolpcu `r_type` nastavimo vrednost `1` in v stolpcu `r_itype` nastavimo vrednost `1` le tistim podatkom, ki imajo izpolnjujejo pogoj `r_itype=0 AND (frc=0 OR freeway=1)`. Torej iz tabele `_ceste` bomo ceste združevali po stolpcu `name`, ki vsebuje imena avtocest in združevali bomo le tiste, ki imajo `r_type=1`, torej avtoceste ceste. Ko imamo združene avtoceste, potem te podatke z funkcijo `izvoziShpDatoteke` pretvorimo podatke iz tabele `_tmp` v imenik `lokacija` **shape datoteko** `_avto_ceste.*`.

```
//Obdelava avto cest za državo.
zacetekAkcije("AKCIJA " + akcija + "-AVTOCESTE DRZAVA " +
drzava + " DISTRIBUCIJA " + dist);

izvediSqlUkaz(psqlGeoDatabase, "DROP TABLE _tmp;");
izvediSqlUkaz(psqlGeoDatabase, "UPDATE _ceste SET r_type = 1,"
+ "r_itype = 1 WHERE r_itype = 0 AND (frc = 0 OR freeway =" +
" 1)");
zdruziLinije(psqlGeoDatabase, "_ceste", "name", "_tmp",
"r_type = 1", null);
izvoziShpDatoteke(psqlGeoDatabase, lokacija +
"\\avto_ceste.shp", "_tmp");

konecAkcije();
```

Z akcijo `obdelava_cest-HITRE CESTE DRZAVA` najprej zaradi varnosti izbrišemo tabelo `_tmp` v katero se bodo z klicem funkcije `zdruziLinije` shranili podatki o imenu in geometriji združenih hitrih cest. V tabeli `_ceste` najprej izvedemo SQL ukaz s katerim na tabeli `_ceste` nastavimo nekaj podatkov. Podatke v tabeli spremenimo tako, da v stolpcu `r_type` nastavimo vrednost 2 in v stolpcu `r_itype` nastavimo vrednost `1` le tistim podatkom, ki imajo izpolnjujejo pogoj `r_itype=0 AND frc IN(1,2)`. Torej iz tabele `_ceste` bomo ceste združevali po stolpcu `name`, ki vsebuje imena hitrih cest in združevali bomo le tiste, ki imajo `r_type`=2, torej hitre ceste. Ko imamo združene hitre ceste, potem te podatke z funkcijo `izvoziShpDatoteke` pretvorimo podatke iz tabele `_tmp` v imenik `lokacija` **shape datoteko** `_hitre_ceste.*`.

```
//Obdelava hitrih cest za državo.
zacetekAkcije("AKCIJA " + akcija + "-HITRE CESTE DRZAVA " +
drzava + " DISTRIBUCIJA " + dist);

izvediSqlUkaz(psqlGeoDatabase, "DROP TABLE _tmp;");
izvediSqlUkaz(psqlGeoDatabase, "UPDATE _ceste SET r_type = 2,"
+ " r_itype = 1 WHERE r_itype = 0 AND frc IN (1,2)");
zdruziLinije(psqlGeoDatabase, "_ceste", "name", "_tmp",
"r_type = 2", null);
izvoziShpDatoteke(psqlGeoDatabase, lokacija +
"\\hitre_ceste.shp", "_tmp");

konecAkcije();
```

Z akcijo `obdelava_cest-REGIONALNE CESTE DRZAVA` najprej zaradi varnosti izbrišemo tabelo `_tmp` v katero se bodo z klicem funkcije `zdruziLinije` shranili podatki o imenu in geometriji združenih regionalnih cest. V tabeli `_ceste` najprej izvedemo SQL ukaz s katerim na tabeli `_ceste` nastavimo nekaj podatkov. Podatke v tabeli spremenimo tako, da v stolpcu `r_type` nastavimo vrednost `3` in v stolpcu `r_itype` nastavimo vrednost `1` le tistim podatkom, ki imajo izpolnjujejo pogoj `r_itype=0 AND frc IN(3)`.Torej iz tabele `_ceste` bomo ceste združevali

po stolpcu `name`, ki vsebuje imena hitrih cest in združevali bomo le tiste, ki imajo `r_type=3`, torej regionalne ceste. Ko imamo združene regionalne ceste, potem te podatke z funkcijo `izvoziShpDatoteke` pretvorimo podatke iz tabele `_tmp` v imenik `lokacija` **shape datoteko** `_regionalne_ceste.shp`.

```
//Obdelava regionalnih cest za državo.
zacetekAkcije("AKCIJA " + akcija + "-REGIONALNE CESTE DRZAVA "
+ drzava + " DISTRIBUCIJA " + dist);

izvediSqlUkaz(psqlGeoDatabase, "DROP TABLE _tmp;");
izvediSqlUkaz(psqlGeoDatabase, "UPDATE _ceste SET r_type = 3,"
+ "r_itype = 1 WHERE r_itype = 0 AND frc IN (3)");
zdruziLinije(psqlGeoDatabase, "_ceste", "name", "_tmp",
"r_type = 3", null);
izvoziShpDatoteke(psqlGeoDatabase, lokacija +
"\\reg_ceste.shp", "_tmp");

konecAkcije();
```

Z akcijo `obdelava_cest-LOKALNE CESTE DRZAVA` najprej zaradi varnosti izbrišemo tabelo `_tmp` v katero se bodo z klicem funkcije `zdruziLinije` shranili podatki o imenu in geometriji združenih lokalnih cest. V tabeli `_ceste` najprej izvedemo SQL ukaz s katerim na tabeli `_ceste` nastavimo nekaj podatkov. Podatke v tabeli spremenimo tako, da v stolpcu `r_type` nastavimo vrednost `4` in v stolpcu `r_itype` nastavimo vrednost `0` le tistim podatkom, ki imajo izpolnjujejo pogoj `r_itype=0 AND frc>3 AND net2class<3`. Torej iz tabele `_ceste` bomo ceste združevali po stolpcu `name`, ki vsebuje imena hitrih cest in združevali bomo le tiste, ki imajo `r_type=4`, torej lokalne ceste. Ko imamo združene lokalne ceste, potem te podatke z funkcijo `izvoziShpDatoteke` pretvorimo podatke iz tabele `_tmp` v imenik `lokacija` **shape datoteko** `_lok_ceste.*`.

```
//Obdelava lokalnih cest za državo.
zacetekAkcije("AKCIJA " + akcija + "-LOKALNE CESTE DRZAVA " +
drzava + " DISTRIBUCIJA " + dist);

izvediSqlUkaz(psqlGeoDatabase, "DROP TABLE _tmp;");
izvediSqlUkaz(psqlGeoDatabase, "UPDATE _ceste SET r_type = 4, "
+"r_itype = 1 WHERE r_itype = 0 AND frc > 3 AND net2class <3");
zdruziLinije(psqlGeoDatabase, "_ceste", "name", "_tmp",
"r_type = 4", null);
izvoziShpDatoteke(psqlGeoDatabase, lokacija +
"\\lok_ceste.shp", "_tmp");

konecAkcije();
```

Sedaj imamo obdelane geografske podatke o cestah potrebno je potrebno še obdelati geografske podatke o cestnih oznakah. Ravno tako kot smo pri cestah ločili podatke po tipu cest bomo sedaj ločili podatke po vrsti cestnih oznakah in pri tem bomo za vsako cestno oznako ustvarili **shape datoteko**, ki bo vsebovala vse geografske podatke o eni cestni oznaki.

Z akcijo `obdelava_cest-OZNAKE CEST DRZAVA` najprej izvedemo SQL ukaze s katerimi na tabeli `_ceste` nastavimo nekaj podatkov. Da bomo lahko geografske podatke v tabeli _ceste ločili po cestnih oznakah bomo ustvarili stolpca poimenovana `r_rtesym` in `r_rteprio`.

Podatke v tabeli spremenimo tako, da v stolpcu `r_rtesym` nastavimo v vrednost `prte` (globalna spremenljivka, ki vsebuje podatke iz tabele `TabDrzava`) in v stolpcu `r_rteprio` nastavimo v vrednost `99` le tistim podatkom, ki imajo v stolpcu `r_itype` vrednost `1` (avtoceste).

```
//OZNAKE CEST ----------------------------------------
//Obdelava cestnih oznak za državo.
zacetekAkcije("AKCIJA " + akcija + "-OZNAKE CEST DRZAVA " +
drzava + " DISTRIBUCIJA " + dist);

izvediSqlUkaz(psqlGeoDatabase,
"ALTER TABLE _ceste DROP COLUMN r_rtesym;");
izvediSqlUkaz(psqlGeoDatabase,
"ALTER TABLE _ceste DROP COLUMN r_rteprio;");
izvediSqlUkaz(psqlGeoDatabase,
"ALTER TABLE _ceste ADD COLUMN r_rtesym text;");
izvediSqlUkaz(psqlGeoDatabase,
"ALTER TABLE _ceste ADD COLUMN r_rteprio int2;");

//Mapiramo rtetyp v simbol in prioriteto glede pravil po državah
var prte = tabDrzav[drzava].privzetZnak;
izvediSqlUkaz(psqlGeoDatabase, "UPDATE _ceste SET r_rtesym = '"
+ prte + "', r_rteprio = 99 WHERE r_itype = 1");
```

Nastavimo podatke še za vse ostale cestne oznake, ki so shranjeni v tabeli `tabDrzava` tako, da izvedemo SQL ukaz `"UPDATE _ceste SET routenum=replace(routenum, '" + zamenjava + "', ''), r_rtesym='" + rtea[i][2] + "', r_rteprio=" + rtea[i][1] + " WHERE r_itype=1 AND rtetyp IN(" + rtea[i][0] + ")".`

```
var rtea = tabDrzav[drzava].znak || [];
for(var i = 0; i < rtea.length; ++i)
{
    var zamenjava = rtea[i][3] || '';
    izvediSqlUkaz(psqlGeoDatabase,
    "UPDATE _ceste SET routenum = replace(routenum, '" +
    zamenjava + "', ''), r_rtesym = '" + rtea[i][2] +
    "', r_rteprio = " + rtea[i][1] +
    " WHERE r_itype = 1 AND rtetyp IN (" + rtea[i][0] + ")");
}
izvediSqlUkaz(psqlGeoDatabase, "DROP TABLE _tmp;");
izvediSqlUkaz(psqlGeoDatabase, "DROP TABLE _tmp2;");
zdruziLinije(psqlGeoDatabase, "_ceste",
"r_rtesym,routenum,r_type,r_rteprio", "_tmp",
"r_itype = 1 AND sliprd = 0 AND rtetyp > 0 AND" +
" length(routenum) > 0", "r_rteprio");
izvediSqlUkaz(psqlGeoDatabase,
"DELETE FROM _tmp WHERE length(the_geom) < 1000;");
zdruziLinije(psqlGeoDatabase, "_tmp",
"r_rtesym,routenum,r_type,r_rteprio", "_tmp2", null,
"r_rteprio");
```

Sedaj, ko imamo podatke o vseh cestnih oznakah urejene, potem z funkcijo `zdruziLinije` shranimo združene podatke iz tabele `_ceste` v tabelo `_tmp` tako, da ima nova tabela `_tmp` podatke, ki so v stolpcih `r_rtesym`, `routenum`, `r_type` in `r_rteprio`. Torej iz tabele `_ceste` bomo združevali ceste po teh stolpcih. V tabeli `_ceste` izvedemo še SQL ukaz s katerim na tabeli `_ceste` zbrišemo podatke, ki imajo dolžino manjšo od `1000m`. Ko so v tabeli `_tmp` združeni podatki jih nato še enkrat z funkcijo `zdruziLinije` združimo in shranimo v drugo tabelo `_tmp2`, ta pa vsebuje enake stolpce kot tabela `_tmp`. Funkcijo `zdruziLinije` smo še enkrat uporabili, zato ker smo iz tabele `_tmp` izbrisali še nekatere nepotrebne podatke.

Sedaj, ko imamo združene podatke, nato te podatke z funkcijo `izvoziShpDatoteke` pretvorimo obdelane podatke iz tabele `_tmp2` v nove **shape datoteke** `_imena_cest_0.*`, `_imena_cest_1.*`, `_imena_cest_2.*`, `_imena_cest_3.*`, `_imena_cest_4.*`.

```
for(var i=0; i<5; ++i)
{
    izvoziShpDatoteke(psqlGeoDatabase, lokacija +
    "\\_imena_cest" + i + ".shp","\"SELECT r_rtesym, routenum,"
    + " the_geom FROM _tmp2 WHERE r_type = " + i +
    " ORDER BY r_rteprio, routenum;\"");
}
```

Potreben je še zadnji postopek obdelave danih geografskih podatkov za Slovenijo in sicer, radi bi imeli poimenovane tudi ceste in jih bomo ravno tako kot pri vseh cestnih obdelavah, imena cest ločili po tipu ceste.

Ker imamo imena cest shranjena v datoteki `imena_cest.dbf`, zato je predenj začnemo z obdelavo podatkov potrebno preveriti ali datoteka `imena_cest.dbf` obstaja v imeniku `lokacija` in ali je velikost datoteke večja od `512bytes`. Če velja pogoj, potem na tabeli `_ceste` izvedemo SQL ukaza s katerima na tej tabeli dodamo stolpca `rtetyp2` in `routenum2`. Ta stolpca dodamo v tabelo `_ceste` zato, ker smo rekli, da bomo cestna imena ločiti in sicer po tipu ceste.

```
//ALTERNATIVNE OZNAKE CEST -----------------------------------
//Sedaj uredimo še alternativne oznake cest (če ostajajo)
if(fso.FileExists(lokacija + "\\imena_cest.dbf") &&
   fso.GetFile(lokacija + "\\imena_cest.dbf").size > 512)
{
    izvediSqlUkaz(psqlGeoDatabase,
    "ALTER TABLE _ceste DROP COLUMN rtetyp2;");
    izvediSqlUkaz(psqlGeoDatabase,
    "ALTER TABLE _ceste DROP COLUMN routenum2;");
    izvediSqlUkaz(psqlGeoDatabase,
    "ALTER TABLE _ceste ADD COLUMN rtetyp2 int2;");
    izvediSqlUkaz(psqlGeoDatabase,
    "ALTER TABLE _ceste ADD COLUMN routenum2 varchar(10);");
```

Datoteko `imena_cest.dbf` je potrebno za obdelavo geografskih podatkov o cestnih imenih za Slovenijo uvoziti v relacijsko bazi `_slovenija`. Ker imamo datoteko `imena_cest.dbf` le v obliki DBF, bomo zato uporabili za uvoz podatkov funkcijo `uvoziDbfDatoteke`, ki omogoča uvoz geografskih podatkov v relacijsko bazo le iz datoteke oblike DBF. V našem primeru bomo podatke iz dane datoteke o cestnih imenih shranili v novo tabelo `_imena`. To storimo zato, ker podatki iz te datoteke nimajo enakih atributov kot jih ima tabela `_ceste`.

```
    izvediSqlUkaz(psqlGeoDatabase, "DROP TABLE _imena;");
    uvoziDbfDatoteke(psqlGeoDatabase, "_imena", lokacija +
    "\\*imena_cest.dbf",kodnaTab);
```

Ker je potrebno cestne oznake ločiti po tipu cest, zato izvedemo SQL ukaze s katerimi nastavimo podatke v tabeli `_ceste` v stolpcih `rtetyp2`, `routenum2` in `r_rtesym` tako, da nastavimo v teh stolpcih vrednosti, kot je prikazano.

```
    izvediSqlUkaz(psqlGeoDatabase,
    "UPDATE _ceste SET rtetyp2 = 0 WHERE r_itype = 1;");
    izvediSqlUkaz(psqlGeoDatabase, "UPDATE _ceste SET rtetyp2="
    + " imena.rtetyp, routenum2 = trim(_rn.routenum) FROM "
    + "_imena WHERE _ceste.id = _imena.id AND r_itype = 1");
    izvediSqlUkaz(psqlGeoDatabase, "UPDATE _ceste SET"
```

```
                + " routenum2='' WHERE routenum=routenum2 AND r_itype= 1");
                izvediSqlUkaz(psqlGeoDatabase, "DROP TABLE _imena;");
```

Vemo, da je `prte` globalna spremenljivka, ki vsebuje podatke iz tabele `TabDrzava` in z drugim SQL ukazom v stolpcu `r_rteprio` nastavimo v vrednost 99 le tistim podatkom, ki imajo v stolpcu `r_itype` vrednost `1`. Potem nastavimo podatke še za vse ostale cestne oznake, ki so shranjeni v tabeli `tabDrzava`. Nastavimo jih tako, da izvedemo SQL ukaz `"UPDATE _ceste SET routenum2=replace(routenum2, '" + zamenjava + "', ''), r_rtesym='" + rtea[i][2] + "', r_rteprio=" + rtea[i][1] + " WHERE r_itype=1 AND rtetyp2 IN(" + rtea[i][0] + ")"`.

```
                izvediSqlUkaz(psqlGeoDatabase,
                "UPDATE _ceste SET r_rtesym = '" + prte +
                "', r_rteprio = 99 WHERE r_itype = 1");

                for(var i=0; i<rtea.length; ++i)
                {
                    var zamenjava = rtea[i][3] || '';
                        izvediSqlUkaz(psqlGeoDatabase,
                        "UPDATE _ceste SET routenum2 =replace(routenum2, '"
                        + zamenjava + "', ''), r_rtesym = '" + rtea[i][2] +
                        "', r_rteprio = " + rtea[i][1] +
                        " WHERE r_itype = 1 AND rtetyp2 IN ("
                        + rtea[i][0] + ")");
                }
```

Nekaj podatkov o cestnih oznakah v tabeli `_ceste` smo sedaj obdelali, potrebno je še z funkcijo `zdruziLinije` združiti nekaj podatkov, saj bomo s tem pospešili iskanje podatkov v tabeli `_tmp`. Torej, enako kot prejšnjem primeru smo uporabili funkcijo `zdruziLinije`, s katero smo iz tabele `_ceste` shranili združene podatke v tabelo `_tmp`. Da bomo še dodatno pospešili iskanje podatkov v tabeli _tmp izbrišemo v njej tiste podatke, ki imajo dolžino manjšo od `1000m` zato ker, teh podatkov ne bomo potrebovali. Potem moramo zaradi učinkovitosti uporabiti na tej tabeli ponovno funkcijo `zdruziLinije`, s katero iz tabele `_tmp` shranimo združene podatke v tabelo `_tmp2`.

```
                izvediSqlUkaz(psqlGeoDatabase, "DROP TABLE _tmp;" +
                "DROP TABLE _tmp2;");
                zdruziLinije(psqlGeoDatabase, "_ceste",
                "r_rtesym,routenum2,r_type,r_rteprio", "_tmp",
                "r_itype = 1 AND sliprd = 0 AND rtetyp2 > 0 AND "
                + "length(routenum2) > 0", "r_rteprio");
                izvediSqlUkaz(psqlGeoDatabase,
                "DELETE FROM _tmp WHERE length(the_geom) < 1000;");
                zdruziLinije(psqlGeoDatabase, "_tmp",
                "r_rtesym,routenum2,r_type,r_rteprio", "_tmp2", null,
                "r_rteprio");
```

Sedaj, ko imamo združene in obdelane podatke o cestnih imenih, potem jih z funkcijo `izvoziShpDatoteke` pretvorimo iz tabele `_tmp2` v nove **shape datoteke** `_imena_cest0.*`, `_imena_cest1.*`, `_imena_cest2.*`, `_imena_cest3.*`, `_imena_cest4.*`.

```
                for(var i = 0; i < 5; ++i)
                {
                    izvoziShpDatoteke(psqlGeoDatabase, lokacija +
                    "\\_imena_cest_1" + i + ".shp",
                    "\"SELECT r_rtesym, routenum2 AS routenum, the_geom "
                    + "FROM _tmp2 WHERE r_type = " + i +
                    " ORDER BY r_rteprio, routenum2;\"");
```

```
                }
                izvediSqlUkaz(psqlGeoDatabase, "DROP TABLE _tmp;" +
                "DROP TABLE _tmp2;");
```

Potrebno je še čiščenje po imeniku tistih datotek, ki so se kopirali pri zagonu funkcije `priskrbiDatoteke`. To storimo s funkcijo `pobriseDatoteke`, ki počisti iz novega imenika željene datoteke.

```
                pobriseDatoteke(lokacija, [shpDbfShx('ceste'),
                'imena_cest.dbf']);
              }

            konecAkcije();
        }

        catch(ex)
        {
            izpis("AKCIJA " + akcija + " DRZAVA " + drzava + " NAPAKA " +
             ex.message);
        }

        konecAkcije();
    }
```

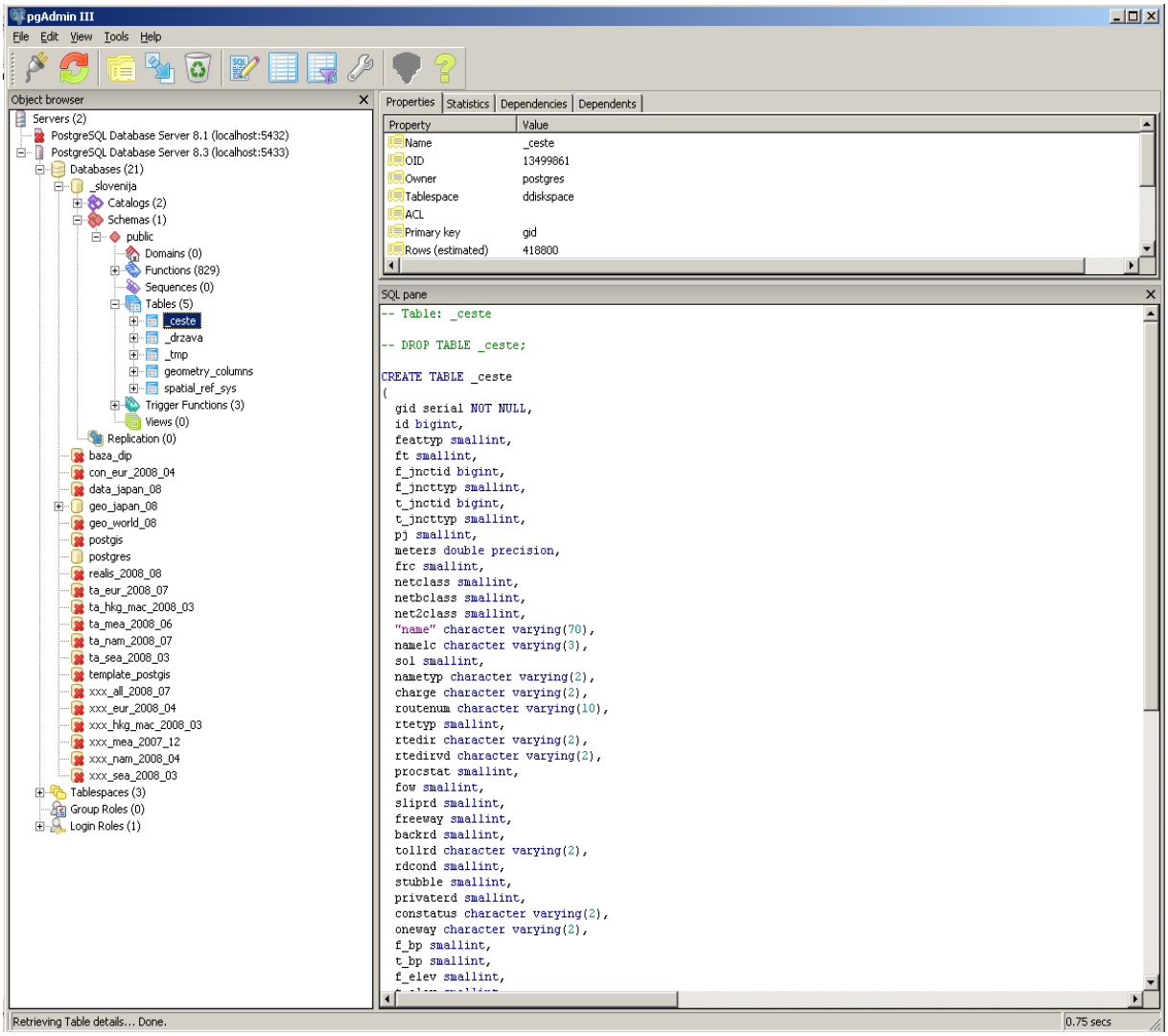
Programsko kodo, s katero obdelamo geografske podatke za Slovenijo, izvršimo v ukazni vrstici z ukazom `cscript JScript3.js JScript2.js obdelava_cest slo`.

Z ukazom v ukazni vrstici smo zagnali programsko kodo, ki je izvršila izbrano akcijo to je `obdelava_cest` za državo Slovenijo.
Z akcijo smo pripravili ceste in njihove oznake za državo Slovenijo tako, da smo:

- s funkcijo `priskrbiDatoteke` kopirali dane podatke o cestah in njihovih oznakah v nov imenik *D:\ADA\DIP_indata\slo2008.07\slo\slo_____* in jih tam razpakirali;
- postopke obdelave geografskih podatkov o cestah in njihovih oznakah razdelili na več delov in jih zaradi preglednosti ločili na akcije:
    - `uvoz`, ki s funkcijo `uvoziShpDatoteke` pretvori geografske podatke iz novega imenika v bazo `_slovenija` in naredi transformacijo koordinat geografskih podatkov iz Lambertove projekcije v transverzalno Mercatorjevo projekcijo;
    - `trajekti`, ki s funkcijo `izvoziShpDatoteke` pretvori obdelane podatke v novo **shape datoteko** `_trajekti`;
    - `euroceste`, ki s funkcijo `izvoziShpDatoteke` pretvori obdelane podatke v novo **shape datoteko** `_eur_ceste`;
    - `avtoceste`, ki s funkcijo `izvoziShpDatoteke` pretvori obdelane podatke v novo **shape datoteko** `_avto_ceste`;
    - `hitre ceste`, ki s funkcijo `izvoziShpDatoteke` pretvori obdelane podatke v novo **shape datoteko** `_hitre_ceste`;
    - `regionalne ceste`, ki s funkcijo `izvoziShpDatoteke` pretvori obdelane podatke v novo **shape datoteko** `_reg_ceste`;
    - `lokalne ceste`, ki s funkcijo `izvoziShpDatoteke` pretvori obdelane podatke v novo **shape datoteko** `_lok_ceste`;
    - `oznake cest`, ki s funkcijo `izvoziShpDatoteke` pretvori obdelane podatke v nove **shape datoteke** `_imena_cest_0`, `_imena_cest_1`, `_imena_cest_2`, `_imena_cest_3`, `_imena_cest0`, `_imena_cest1`, `_imena_cest2`, `_imena_cest3`, `_imena_cest4`;

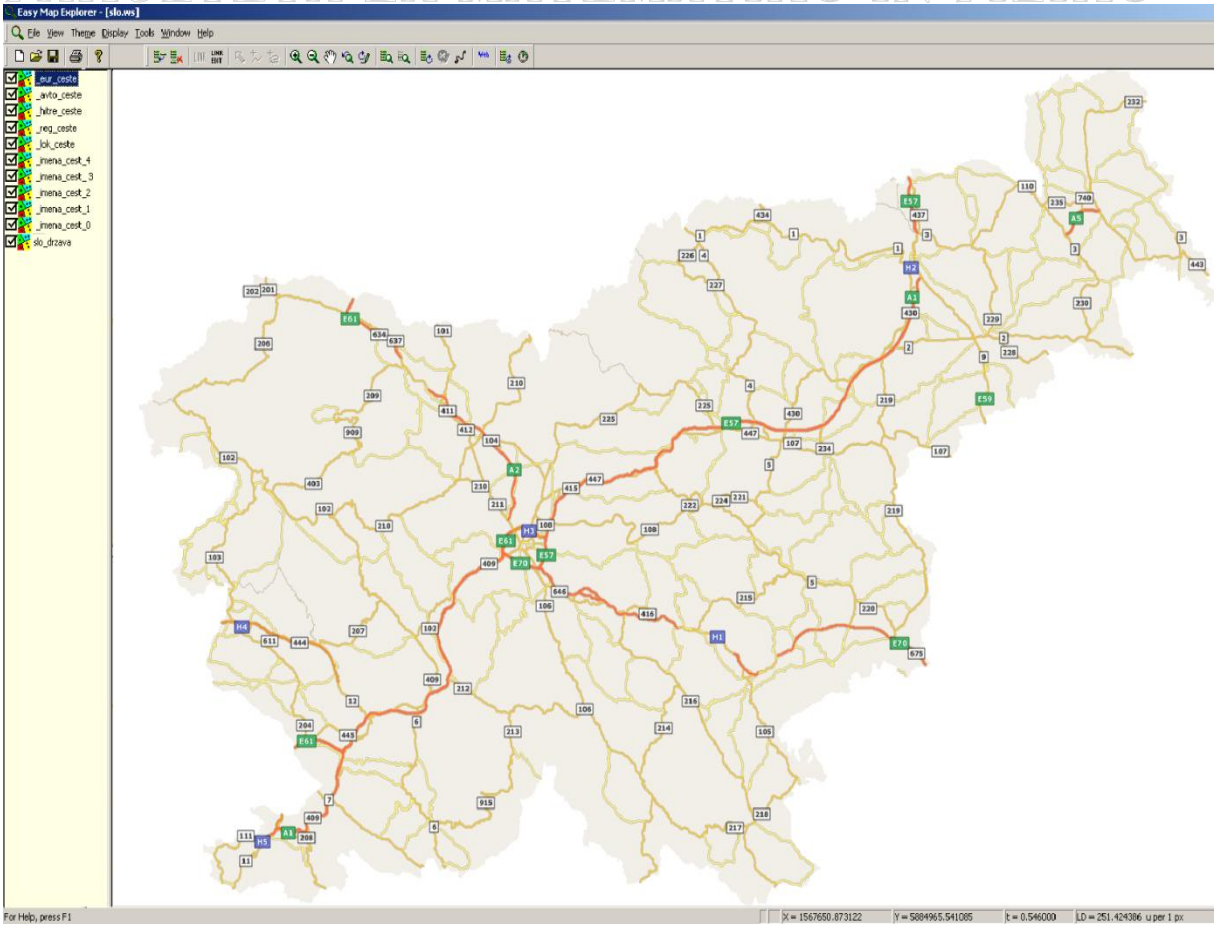
- s funkcijo `pobriseDatoteke` počistili iz novega imenika tiste podatke, ki so se kopirali pri zagonu funkcije `priskrbiDatoteke`.

Ker je rezultat zelo obširen, si bomo pogledali rezultat v PostgreSQL-u (Slika 34). Vidimo, da se je v relacijski bazi `_slovenija` ustvarila nova tabela `_ceste`, ki je napolnjena z geografskim podatki tipa `MultiLineString`.

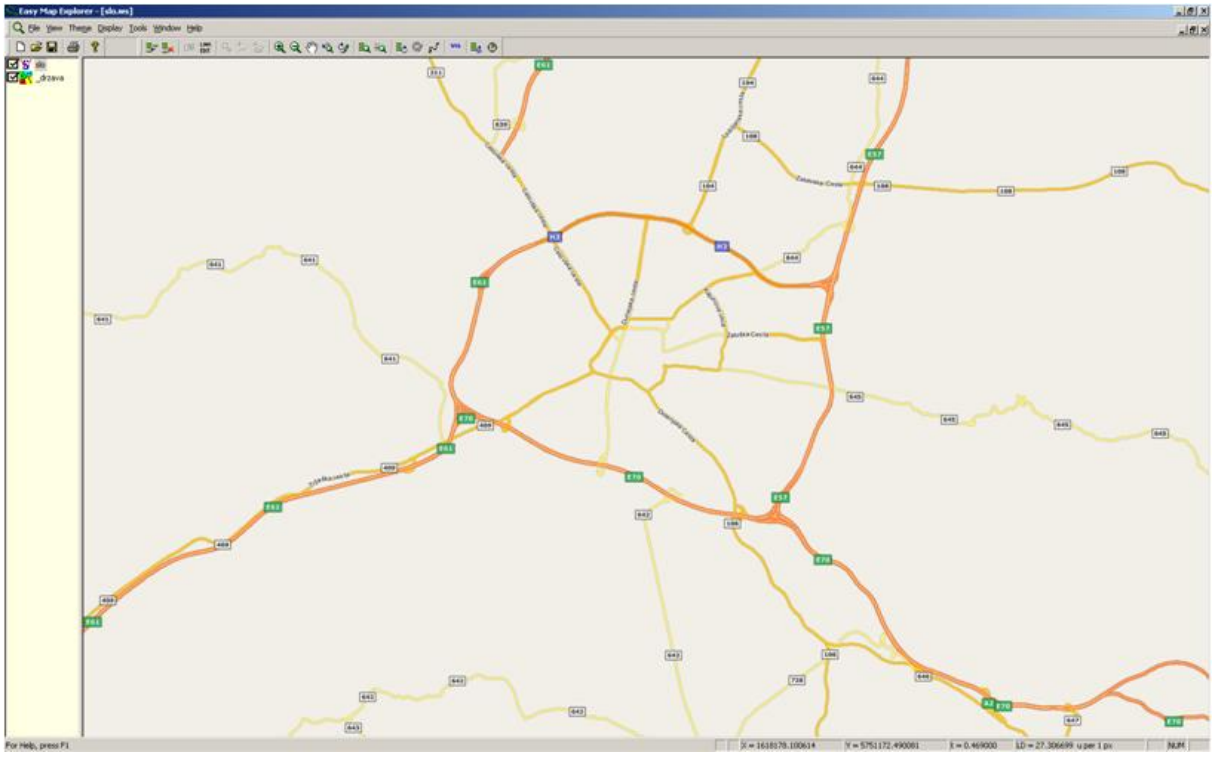


Slika 34 Prikaz zgradbe tabele _ceste v relacijski bazi _slovenija

Geografski podatki, ki smo jih pridobili z izvršeno programsko kodo, se nahajajo v imeniku *D:\ADA\DIP_outData\slo*.** Poglejmo si vso vsebino geografskih podatkov za Slovenijo z GIS orodjem **Easy Map Explorer** (Slika 35).



Slika 35 Prikaz ustvarjenih datotek v GIS orodju Easy Map Explorer



Slika 36 Prikaz ustvarjenih datotek v GIS orodju Easy Map Explorer

ZAHVALA

Iskreno se zahvaljujem mag. Matiju Lokarju za pomoč in vodenje pri pisanju diplomske naloge.

Zahvaljujem se tudi vsem domačim in prijateljem, ki so mi v času študija stali ob strani. Še posebej gre moja zahvala bratu Alojzu, ki mi je vedno znal svetovati o geografskih informacijskih sistemih.

Za izdelavo diplome pa se izredno zahvaljujem sodelavcem podjetja Realis d.o.o.

Hvala vsem za razumevanje in za potrpežljivost.

SKLEPNA BESEDA

Predstavila sem delo z geografskimi podatki v bazi PostgreSQL z dodatkom PostGIS. Ker PostgreSQL ne omogoča hranjenja geografskih podatkov, sem predstavila razširitev sistema PostgreSQL PostGIS, s katerim pridobimo podatkovne tipe, ki jih sam PostgreSQL ne pozna.
Z dodatkom PostGIS smo dobili tudi dve tabeli, ki se ustvarita v vsaki novo zgrajeni relacijski bazi. To sta tabeli spatial_ref_sys in geometry_columns. Za nas je še posebej pomembna tabela spatial_ref_sys, s katero pridobimo podatke o koordinatnih sistemih. Ker tabela ni vsebovala podatkov o transverzalni Mercatorjevi projekciji, smo le te dodali z ustreznim SQL ukazom.
Da lahko analiziramo ali obdelamo lastnosti geografskih podatkov v bazi, nam PostGIS omogoča tudi dva programa, s katerima pretvorimo lastnosti geografskih podatkov v PostgreSQL bazo in obratno. To sta shp2pgsql.exe in pgsql2shp.exe. S prvim programom pretvorimo lastnosti geografskih podatkov v relacijsko bazo, z drugim pa pretvorimo podatke iz tabele v datoteke oblike SHP, DBF in SHX. Pri pretvarjanju lastnosti geografskih podatkov so lahko tabele zelo velike in je zaradi tega poizvedba po tabelah zelo počasna. Zato nam PostGIS omogoča uporabo GIST indeksov, ki so osnovna oblika indeksiranja GIS podatkov, ki pa jih sam PostgreSQL ne pozna.

Glavni namen diplomske naloge je bil ugotoviti, kako s tehnologijami PostgreSQL, PostGIS in JScript ter z geografskimi podatki izdelati karto Slovenije.

Priporočam uporabo vseh navedenih tehnologij, ki sem jih opisala v diplomski nalogi, vsem, ki delajo z GIS-om.

LITERATURA

Radoš Šumrada, 2005: *Tehnologija GIS* . Ljubljana: Fakulteta za gradbeništvo in geodezijo.

Spletni viri

- Dokumentacija za PostgreSQL, dostopno na naslovu
  http://www.postgresql.org/docs/
  (zadnji obisk: 18. 1. 2009)
- Dokumentacija za postGIS, dostopno na naslovu
  http://postgis.refractions.net/documentation/
  (zadnji obisk: 18. 1. 2009)
- Drobne S.,*GIS pojmi*, dostopno na naslovu
  http://www.fgg.uni-lj.si/~/sdrobne/GIS_Pojm/Index.htm
  (zadnji obisk: 18. 1. 2009)
- Strobl C. Dimensionally Extended Nine-Intersection Model (DE-9IM), dostopno na naslovu
  http://www.gis.hsr.ch/wiki/images/3/3d/9dem_springer.pdf
  (zadnji obisk: 18. 1 .2009)
- Point Set Theory and the DE-9IM Matrix, dostopno na naslovu
  http://docs.codehaus.org/display/GEOTDOC/Point+Set+Theory+and+the+DE-9IM+Matrix
  (zadnji obisk: 18.1. 2009)
- EPSG Projection 4326 - WGS 84 , dostopno na naslovu
  http://spatialreference.org/ref/epsg/4326/
  (zadnji obisk: 18. 1. 2009)
- ESRI Projection 54004 – world mercator, dostopno na naslovu
  http://spatialreference.org/ref/esri/54004/
  (zadnji obisk: 18. 1. 2009)
- OpenGIS® Implementation Specification for Geographic
  Information, Simple feature access - Part 1:Common architecture, dostopno na naslovu
  http://www.opengeospatial.org/standards/sfa
  (zadnji obisk: 18. 1. 2009)
- Wikipedia, Sferni koordinatni sistem, dostopno na naslovu
  http://sl.wikipedia.org/wiki/Sferni_koordinatni_sistem
  (zadnji obisk: 18. 1. 2009)
- Wikipedia, B-Tree, dostopno na naslovu
  http://en.wikipedia.org/wiki/B-tree
  (zadnji obisk: 18. 1. 2009)
- Wikipedia, R-Tree, dostopno na naslovu
  http://en.wikipedia.org/wiki/R-tree
  (zadnji obisk: 18. 1. 2009)
- Wikipedia, GIST, dostopno na naslovu
  http://en.wikipedia.org/wiki/GiST
  (zadnji obisk: 18. 1. 2009)

- MySQL, The OpenGIS Geometry Model, dostopno na naslovu
  http://dev.mysql.com/doc/refman/5.0/en/opengis-geometry-model.html
  (zadnji obisk: 18. 1. 2009)

- Manifold, Dissolve, dostopno na naslovu
  http://www.manifold.net/doc/dissolve.htm
  (zadnji obisk: 18.,1. 2009)

- Krevs M., Repe B., Geografski informacijski sistemi, dostopno na
  http://www.zrss.si/ppt/GEO_multplik_b.ppt
  (zadnji obisk: 18. 1. 2009)

- Geoservis, Geografski informacijski sistemi, dostopno na
  http://www.geoservis.si/uporabno/gis/gis.htm
  (zadnji obisk: 18. 1. 2009)

- Šinigoj j., Komac M., Geološka karta v GIS okolju, dostopno na
  http://www.geo-zs.si/publikacije_arhiv/Clanki/Geologija_45_2/sinigoj_etal_45_2.pdf
  (zadnji obisk: 18. 1. 2009)

- Microsoft, JScript, dostopno na
  http://msdn.microsoft.com/en-us/library/hbxc2t98.aspx
  (zadnji obisk: 18. 1. 2009)

- W3schools, JScript, dostopno na
  http://www.w3schools.com/jS/js_intro.asp
  (zadnji obisk: 18. 1. 2009)

- Wikipedia, JScript in Windows Script Host, dostopno na
  http://en.wikipedia.org/wiki/JScript#JScript_in_Windows_Script_Host
  (zadnji obisk: 18. 1. 2009)