VYSOKÁ ŠKOLA BÁŇSKÁ – TU OSTRAVA
Hornicko-geologická fakulta
Katedra geoinformatiky

ANALÝZA MOŽNOSTÍ SŘBD PostgreSQL A NÁDSTAVBY PostGIS PRO VYTVOŘENÍ DATOVÉHO SKLADU V PROSTŘEDÍ GIS

Diplomová práce

Autor: Bc. Jaromír Kamler
Vedoucí bakalářské práce: Ing. Antonín Orlík

Ostrava 2006

Prohlášení

- *Celou diplomovou práci včetně příloh, jsem vypracoval samostatně a uvedl jsem všechny použité podklady a literaturu.*
- *Jsem byl seznámen s tím, že na moji diplomovou práci se plně vztahuje zákon č.121/2000 Sb. - autorský zákon, zejména § 35 – využití díla v rámci občanských a náboženských obřadů, v rámci školních představení a využití díla školního a § 60 – školní dílo.*
- *Beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, diplomovou práci užít (§ 35 odst. 3).*
- *Souhlasím s tím, že jeden výtisk diplomové práce bude uložen v Ústřední knihovně VŠB-TUO k prezenčnímu nahlédnutí a jeden výtisk bude uložen u vedoucího diplomové práce. Souhlasím s tím, že údaje o diplomové práci, obsažené v Záznamu o závěrečné práci, umístěném v příloze mé diplomové (resp. bakalářské) práce, budou zveřejněny v informačním systému VŠB-TUO.*
- *Rovněž souhlasím s tím, že kompletní text diplomové práce bude publikován v materiálech zajišťujících propagaci VŠB-TUO, vč. příloh časopisů, sborníků z konferencí, seminářů apod. Publikování textu práce bude provedeno v omezeném rozlišení, které bude vhodné pouze pro čtení a neumožní tedy případnou transformaci textu a dalších součástí práce do podoby potřebné pro jejich další elektronické zpracování.*
- *Bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst.4 autorského zákona.*
- *Bylo sjednáno, že užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).*

V Ostravě dne 10.4.2006 ***Jaromír Kamler***

……

Jaromír Kamer
Husova 580/8
Jeseník

Anotace

Práce se zabývá analýzou možností centralizovaného ukládání prostorových dat do SŘBD PostgreSQL s extensí PostGIS definující nové funkce a datové typy pro ukládání geodat. Jsou popsány možnosti automatické správy databáze pomocí tvorby vlastních funkcí a triggerů nebo pomocí software. Dále je popsáno použití indexace pro efektivní práci s databází a jsou uvedeny některé základní funkce včetně příkladů pracující s prostorovými daty. Práce se také věnuje některým nejrozšířenějším nástrojům schopným přistupovat k datům uloženým v PostgreSQL/PostGIS.

Klíčová slova: PostgreSQL, PostGIS, databáze, geodata

Annotation:

My work deals with the interpretation of the possibility of centralized space data stacking into RDBMS PostgreSQL with the extension of PostGIS defining new functions and data types for the storing of geodetic data. There is described potential of the automatic control of database through the generation of the actual functions and triggers or via software. In the following is described the use of data indexing on behalf of effective operation with the database and there are also mentioned some of the basic functions including the examples working with the space data. My work also pursues some of the most expanded tools which are able to access data stored in PostgreSQL/PostGIS.

Key words: PostgreSQL, PostGIS, database, geodata

Obsah

Seznam použitých zkratek

CGI	Common Gateway Interface
GIS	Geographic Information Systems
GiST	Generalized Search Tree
GNU	Rekurzivní zkratka pro GNU's Not Unix. Projekt zaměřený na svobodný software inspirovaný operačními systémy unixového typu
GPL	General Public Licence
HTTP	Hyper Text Transfer Protocol
NIS	Network Information Service
OS	Operační Systém
PL	Procedural Languages
RDBMS	Relational Database Management System
SQL	Structured Query Language
SŘBD	Systém Řízení Báze Dat
SSL	Secure Sockets Layer. Protokol pro bezpečnou komunikaci na Internetu poskytující šifrování a ověřování pravosti transakcí.
SVG	Scalable Vector Graphics
UDF	User defined functions
WFS	Web Feature Service
WKB	Well Known Binary. Binární formát ukládání geodat v PostgreSQL/PostGIS.
WKT	Well Known Text. Textový formát ukládání geodat v PostgreSQL/PostGIS.
WMS	Web Map Service

1 ÚVOD

Každý objekt nebo jev je možné jednoznačně identifikovat pomocí jeho popisných a prostorových atributů. Soubor těchto atributů je označován jako prostorová data. Prostorová data je možné v prostředí GIS zaznamenávat buď ukládáním do souborů, nebo do relačních databází.V případě nutnosti ukládání velkého množství prostorových dat a řízení přístupu k nim pro větší množství uživatelů je vhodné i přes některé výhody klasického souborového ukládání dat používat relační databáze. Relační databáze umožňují využívat velkou řadu výhod, které tento způsob uložení dat přináší. RDBMS poskytuje širokou škálu nástrojů pro správu dat, stejně tak i nástroje pro jejich analýzu.

Výhodou použití relační databáze pro ukládání prostorových dat je mimo jiné i to, že umožňuje ukládat všechna atributová i vektorová geografická data na jednom místě v jedné tabulce oproti například rozšířenému ***shapefile***, kde je nutno mít k uložení dat minimálně tři soubory se zvlášť oddělenými popisnými a grafickými atributy, díky čemuž se správa dat stává přehlednější. Při uložení dat do RDBMS je možné poskytovat data několika různým uživatelům naráz. Další výhodou ukládání velmi objemných dat do RDBMS je skutečnost, že nejsou kladena žádná omezení na maximální velikost tabulky ani databáze.

Ačkoliv není ukládání prostorových dat do relační databáze natolik rozšířené jako klasické souborové, má nesporné výhody. Zavedením relační databáze pro ukládání prostorových a atributových dat bude mít pozitivní vliv především na bezpečnost, přehlednost a jednoduchost poskytování dat určeným uživatelům. Další výhodou je možnost využívání funkcí definovaných na úrovni databáze k práci s prostorovými daty.

2 CÍLE PRÁCE

Cílem práce je analyzovat možnosti využití SŘBD PostgreSQL k ukládání prostorových dat do datového skladu. Je nutné popsat funkcionalitu SŘBD PostgreSQL vzhledem ke specifickým požadavkům, které jsou při tvorbě datového skladu na tuto databázi kladeny.

Hlavními cíli, kterými je nutné se zabývat jsou instalace, konfigurace a zabezpečení přístupu k databázi, import a export dat do databáze, transformace mezi souřadnicovými systémy, popis klientů pomocí nichž je možné se připojit k datovému skladu, jejich funkce, ověřit možnosti správy uživatelů a uživatelských skupin včetně nastavování přístupových práv k jednotlivým tabulkám případně jen určitým záznamům. Dále popsat některé funkce pro práci s prostorovou složkou dat, tvorbu vlastních funkcí a triggerů. Posledním, ale velmi významným cílem z hlediska výkonu SŘBD při prostorových dotazech je porovnání výkonnosti s použitou GiST indexací nebo bez ní.

Dále je nutné zejména v případech importu a exportu dat do databáze a při jejich transformacích mezi kartografickými zobrazeními ověřit, zda jsou tato data identická s původními.

3 SŘBD UMOŽŇUJÍCÍ UKLÁDÁNÍ PROSTOROVÝCH DAT

Většina SŘBD určených pro ukládání a práci s prostorovými daty splňuje standardy konsorcia OGC, ISO a dodržují standardy SQL92 a SQL99. Rozdělení těchto databází je možné provést dle jejich ceny, výkonnosti a funkcí, které poskytují. Dále se zabývám rozdělením podle ceny na komerční a open-source produkty.

3.1 Komerční databáze pro ukládání prostorových dat

Mezi komerční databáze schopné ukládat prostorová data patří především IBM INFORMIX Datablade Technology a Oracle.

U databáze Informix jsou jednotlivá rozšíření pro práci s prostorovými daty realizována pomocí modulů zvaných DataBlade, které jsou zahrnovány přímo do jádra databáze. Databáze Informix disponuje moduly určenými například pro zpracování a správu geoprostorových dat (Informix Spatial Datablade) nebo pro práci s časovými řadami (TimeSeries Datablade).

Databáze Oracle mají již v nejnižší verzi Oracle Database Standard Edition One obsažený Oracle Locator, který umožňuje ukládání prostorových dat a základní dotazy. V případě, že požadujeme např. i převody mezi souřadnicovými systémy, generování nových objektů nebo výpočty ploch či délek je nutné doinstalovat Oracle Spatial.

Ceny licencí na tyto databáze se pohybují řádově ve stovkách tisíc korun za jeden procesor. Konkrétně u Oracle Database Standard Edition One je cena 124.685 Kč/procesor (k 20.2.06) a součástí instalace je jen Oracle Locator, který neobsahuje dostatečné funkce pro práci s prostorovými daty.

3.2 Ukládání prostorových dat pomocí open-source produktů

Hlavní a nespornou výhodou open-source produktů je jejich nulová pořizovací cena, přičemž se ovšem jedná o kvalitní software vyvíjený velmi rozsáhlým týmem zkušených programátorů, kteří jsou roztroušeni po celém světě. Kvalita open-source programů je dána velkým počtem uživatelů, kteří svým každodenním užíváním daného software odhalují chyby. Hlášení o nich jsou shromažďovány na jednom místě a vývojáři odpovědní za danou oblast jsou díky svému rozptýlení po Zemi schopni opravit odhalené chyby do 24 hodin.

Často bývá vývoj takovýchto produktů sponzorován velkými firmami jako jsou například Sun, IBM, Red Hat, … .

Mezi nejvýznamnější open-source databázové projekty patří MySQL a PostgreSQL. Oba dva RDBMS umožňující ukládání prostorových dat podle standardů OGC. PostgreSQL to umožňuje pomocí extense PostGIS zatímco MySQL má tyto vlastnosti implementovány přímo v sobě.

Výkonnostně je na tom PostgreSQL podobně jako další komerční, ale i open source databáze, v některých funkcích je rychlejší, jindy pomalejší. V porovnání s MySQL a podobnými databázovými systémy je PostgreSQL rychlejší při víceuživatelském přístupu, složitějších dotazech a zatížení read/write dotazy. MySQL je rychlejší v jednodušších dotazech s malým počtem uživatelů.[2]

4 PostgreSQL A PostGIS

PostgreSQL je objektově-relační databáze založená na systému Postgres, který byl vyvíjen na University of California at Berkeley Computer Science Department. Postgres se dá označit za průkopníka mnoha konceptů, které se později staly dostupnými i v komerčních databázových systémech. PostgreSQL je open-source produkt pod licencí BSD vycházející přímo z Postgresu. Podporuje standardy SQL92, SQL99 a je možné jej uživatelem rozšiřovat v mnoha směrech.

Mezi tato rozšíření patří i PostGIS uvolněný pod licencí GNU GPL, který umožňuje ukládat do databáze i prostorové objekty běžně využívané v GIS (bod, linie, polygon) ve formě splňující standardy konsorcia OGC. PostGIS dále nabízí celou řadu rozšiřujících funkcí, pomocí nichž je možná poměrně jednoduchá správa a manipulace s těmito objekty. Jedná se např. o funkce typu boolean (Equals, Disjoint, Touches, Within, Overlaps, Crosses, Intersects, Contains, …) nebo constructive (Intersection, Difference, Union, SymDifference, Buffer, ConvexHull, …) a řadu dalších funkcí včetně výstupu do SVG.

4.1 Vlastnosti PostgreSQL

Základní vlastnosti PostgreSQL:

- **Relační** - základní vlastnost všech SQL serverů, díky této vlastnosti jsou databáze mezi různými SQL servery slučitelné. PostgreSQL poskytuje vlastnosti, které jsou standardem mezi SQL servery (transakce, optimalizace dotazů, víceuživatelská podpora atd.);
- **Vysoce rozšiřitelný:**
  - komplexní dotazy;
  - foreign keys (cizí klíče);
  - triggery (spouště);
  - views (pohledy);
  - transakce;
  - vlastní datové typy;
  - agregační funkce;
  - stored procedures (uložené procedury a to nejen ve vlastním PL/SQL).
- umožňuje uživatelem definovant typy, operátory, funkce a přístupové metody;

- **Objektově relační** - databáze má některé objektově orientované vlastnosti jako například dědičnost.

V pgsql běží tyto základní procesy:

- **postmaster** je proces, který řídí komunikaci mezi procesy v pozadí a v popředí a zajišťuje inicializaci;
- **postgres** je proces, který vykonává SQL dotazy;
- Aplikace připojující se k pgsql např. **psql** vysílající požadavky na server.

Pgsql nabízí tato aplikační rozhraní:

- **SQL** - v pgsql je implementována rozšířená podmnožina ANSI SQL;
- **C API** - v knihovně libpq je množina funkcí umožňující přístup k pgsql funkcemi jazyka C;
- **C++ API** - v knihovně libpq++ je množina tříd umožňující přístup k pgsql pomocí jazyka C++;
- **Tcl API** - v knihovně libpgtcl je množina funkcí umožňující přístup k pgsql funkcemi jazyka tcl;
- **Perl API** - pgsql_perl5 je knihovna pro rozhraní Perlu 5;
- **Python API** - PyGres95 je knihovna pro rozhraní Python.

Tab. 1. Velikostní omezení databáze PostgreSQL.

Maximální velikost databáze	neomezena
Maximální velikost tabulky	32 TB
Maximální velikost řádky	1,6 TB
Maximální velikost položky	1 GB
Maximální počet řádků v tabulce	neomezeno
Maximální počet sloupců v tabulce	250-1600 podle typů
Maximální počet indexů na tabulce	neomezeno

Velikostní omezení řádku, tabulky a databáze je uvedeno v tab. 1. Maximální velikost tabulky je 32 TB a nevyžaduje podporu velkých souborů operačním systémem. Velké tabulky se ukládají do několika 1 GB souborů, takže limity souborového systému nejsou podstatné.

5 GIS OBJEKTY, JEJICH IMPORT, EXPORT A PRÁCE S NIMI

GIS objekty jsou definovány OpenGIS specifikacemi, které definují dva standardní způsoby zápisu prostorových objektů: well-known text reprezentaci (WKT) a well-known binary reprezentaci (WKB). Oba dva, WKT i WKB, zahrnují informace o typu objektu a souřadnic, ze kterých je objekt tvořen.

OpenGIS specifikace definují 3 základní typy prostorových dat:

1. Bod, multibod

- POINT(0 0 0)
- MULTIPOINT(0 0 0,1 2 1)

2. Linie, řetězec linií

- LINESTRING(0 0,1 1,1 2)
- MULTILINESTRING((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))

3. Polygon, multipolygon

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

5.1 Referenční systémy

Při importu prostorových dat do databáze je žádoucí těmto datům přiřadit určitý referenční systém, v jakém byla pořízena. Toho se poté dá využívat k transformacím mezi projekcemi.

V databázi jsou vytvořeny dvě tabulky *spatial_ref_sys* a *geometry_columns*.

Definice tabulky spatial_ref_sys:

```
CREATE TABLE SPATIAL REF SYS (
SRID INTEGER NOT NULL PRIMARY KEY,
AUTH_NAME VARCHAR(256),
AUTH_SRID INTEGER,
SRTEXT VARCHAR(2048),
PROJ4TEXT VARCHAR(2048));
```

Sloupce v tabulce SPATIAL_REF_SYS jsou následující:

- SRID - Jedinečný číselný identifikátor prostorového srovnávacího souboru uvnitř databáze.
- AUTH_NAME - Jméno standardu citované pro srovnávací soubor. Například ”EPSG”
- AUTH_SRID - Identifikátor z prostorového srovnávacího souboru.

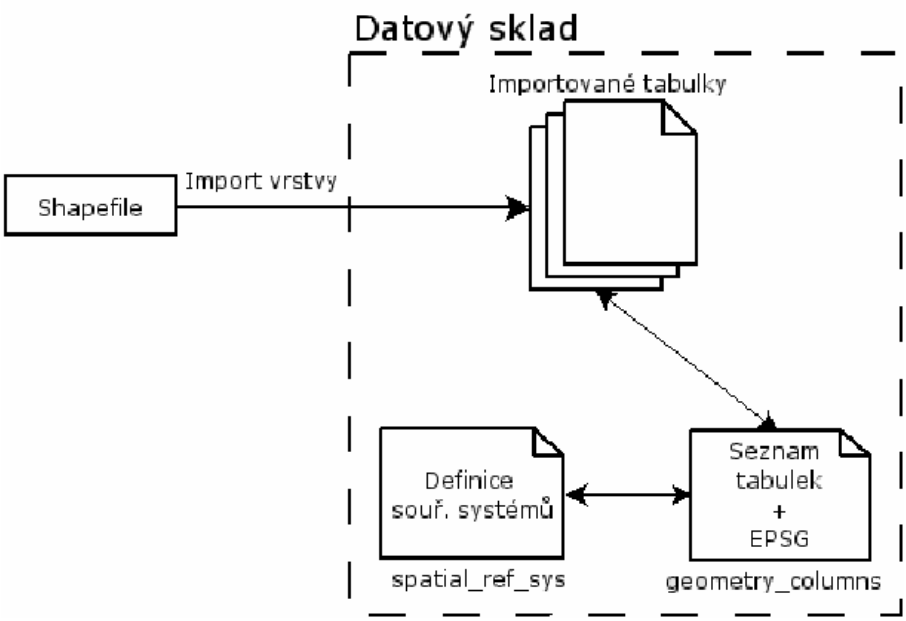
- SRTEXT - Well-known text z prostorového srovnávacího souboru.
- PROJ4TEXT - PostGIS používá Proj4 knihovnu, která poskytuje možnosti pro transformaci souřadnic.

Definice tabulky geometry_columns:

```
CREATE TABLE GEOMETRY_COLUMNS (
F_TABLE CATALOG VARCHAR(256) NOT NULL,
F_TABLE SCHEMA VARCHAR(256) NOT NULL,
F_TABLE NAME 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,);
```

Sloupce v GEOMETRY COLUMNS jsou následující:

- F_TABLE_CATALOG, F_TABLE_SCHEMA, F_TABLE_NAME – sloupce s názvy katalogu, schématu a názvu tabulky.
- F_GEOMETRY_COLUMN – jméno sloupce obsahujícího prostorová data
- COORD_DIMENSION - prostorová dimenze sloupce (2 nebo 3 dimenze).
- SRID - identifikátor prostorového srovnávacího souboru . SRID je klíč odkazující se na tabulku SPATIAL_REF_SYS
- TYPE - typ prostorového objektu.



Obr. 1. Schéma přiřazení SRID importované vrstvě.

Pokud je při importu dat do databáze tabulce přidělen SRID kód (spatial referencing identifier), který představuje určitý referenční systém, jehož definici lze nalézt v tabulce *spatial_ref_sys*, bude tato tabulka zapsána do tabulky *geometry_columns* a bude s ní možné provádět transformace mezi různými projekcemi. Způsob přiřazení souřadnicového systému importované vrstvě je schematicky znázorněn na obrázku 1.

Následujícím dotazem můžeme zjistit, jaké SRID bylo přiděleno při importu tabulce *uliceporuba102065*.

```
SELECT srid(the_geom) FROM uliceporuba102065 limit 1;
srid
----
102065
```

SRID kód 102065 představuje souřadnicový systém S-JTSK. Jeho definici můžeme získat následujícím dotazem v tabulce s*patial_ref_sys*:

```
SELECT proj4text FROM spatial_ref_sys WHERE srid=102065;
proj4text
---------
+proj=krovak +lat_0=49.5 +lon_0=24.83333333
333333 +alpha=30.28813975277778 +k=0.9999 +x_0=0 +y_0=0
+ellps=bessel +units=m no_defs
```

Následujícím dotazem získáme výpis prvního záznamu souřadnic bodů představujících linii:

```
SELECT AsText(the_geom) from uliceporuba102065 limit 1;
AsText
MULTILINESTRING((-479814.40625 -1101977.125,-479818.017232143
-1102000))
```

Pro převod těchto souřadnic do souřadného systému WGS-84, jehož SRID kód je 4326 je možné využít funkce *Transform()*:

```
SELECT AsText(Transform(the_geom,4326)) FROM uliceporuba102065 limit
1;
AsText
MULTILINESTRING((18.15712128448 49.8259056388761,18.1570989730251
49.8256979106819))
```

5.2 Vkládání prostorových dat pomocí SQL, shp2pgsql a ogr2ogr

Při vkládání prostorových dat pomocí SQL je nejprve nutné vytvořit tabulku, v níž bude sloupec s názvem *the_geom* a bude datového typu geometry. Tento datový typ v sobě zahrnuje všechny základní typy prostorových dat. Příklad vytvoření nové tabulky a následné vložení dat uvádím v následujícím výpisu. Na obr. 2 jsou zobrazeny takto vytvořené objekty v JUMP GIS.

```
CREATE TABLE MySpatialTable (nazev varchar(10), the_geom geometry);
```

```
INSERT INTO MySpatialTable (nazev, the_geom) VALUES
('bod', GeometryFromText('POINT(5 5)'));
```

```
INSERT INTO MySpatialTable (nazev, the_geom) VALUES
('obdelnik', GeometryFromText('LINESTRING(1 1, 1 7, 7 7, 7 1, 1
1)'));
```

```
INSERT INTO MySpatialTable (nazev, the_geom) VALUES
('trojuhel', GeometryFromText('POLYGON((0 0 0,4 0 0,4 4 0,0 4 0,0 0
0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))'));
```

Do databáze lze také vkládat prostorová data z *shapefile* pomocí programu *shp2pgsql*. Pomocí parametru -s zadáme SRID kód, který má být datům přiřazen, zdrojový SHP soubor, dále název tabulky a databáze. Výstup z programu *shp2pgsql* poté předáme rourou terminálu psql, ke kterému se připojíme s parametrem -d udávajícím název databáze. K úspěšnému importu dat je ještě nutné zadat přístupové heslo k databázi. Dále uvádím popsaný příkaz.

```
bash-3.00$ /usr/local/pgsql/bin/shp2pgsql -s 102065 /data/ulice.shp
ulicePoruba kam038 | psql -d kam038
```

V případě, že byl při importu *shapefile* do databáze uveden kód prostorového referenčního systému, je tento kód zaznamenán společně s názvem tabulky do tabulky *geometry_columns*. Budeme-li jej chtít exportovat ven z databáze, je to možné pomocí programu *pgsql2shp*. Dále uvádím příklad exportu tabulky *uliceporuba* do *shapefile*.

```
bash-3.00$ /usr/local/pgsql/bin/pgsql2shp -f /data/uliceporuba.shp -
h localhost -P kam038 -u kam038 kam038 uliceporuba
```

Za parametrem -f je název nového souboru, parametr -h je host, -P je heslo, -u název uživatele a dále následuje název databáze a tabulky.

Další možností, jak do databáze importovat data je použít utilitu ogr2ogr. Ogr2ogr je součástí knihovny OGR, která je obsažena v GDAL (Geospatial Data Abstraction Library). Z internetové adresy [14] je možné stáhnout balík FWTools pro GNU/Linux i pro MS Windows obsahující zmíněnou knihovnu včetně dalších knihoven schopných provádět např. transformace mezi souřadnicovými systémy. Knihovna OGR umí v současné době pracovat s 25 druhy datových formátů jako jsou např. ESRI Shapefile, DWG, DGN a další.

Níže uvádím ukázku použití příkazu ogr2ogr při importu dat do databáze PostgreSQL/PostGIS ze souboru ESRI Shapefile :

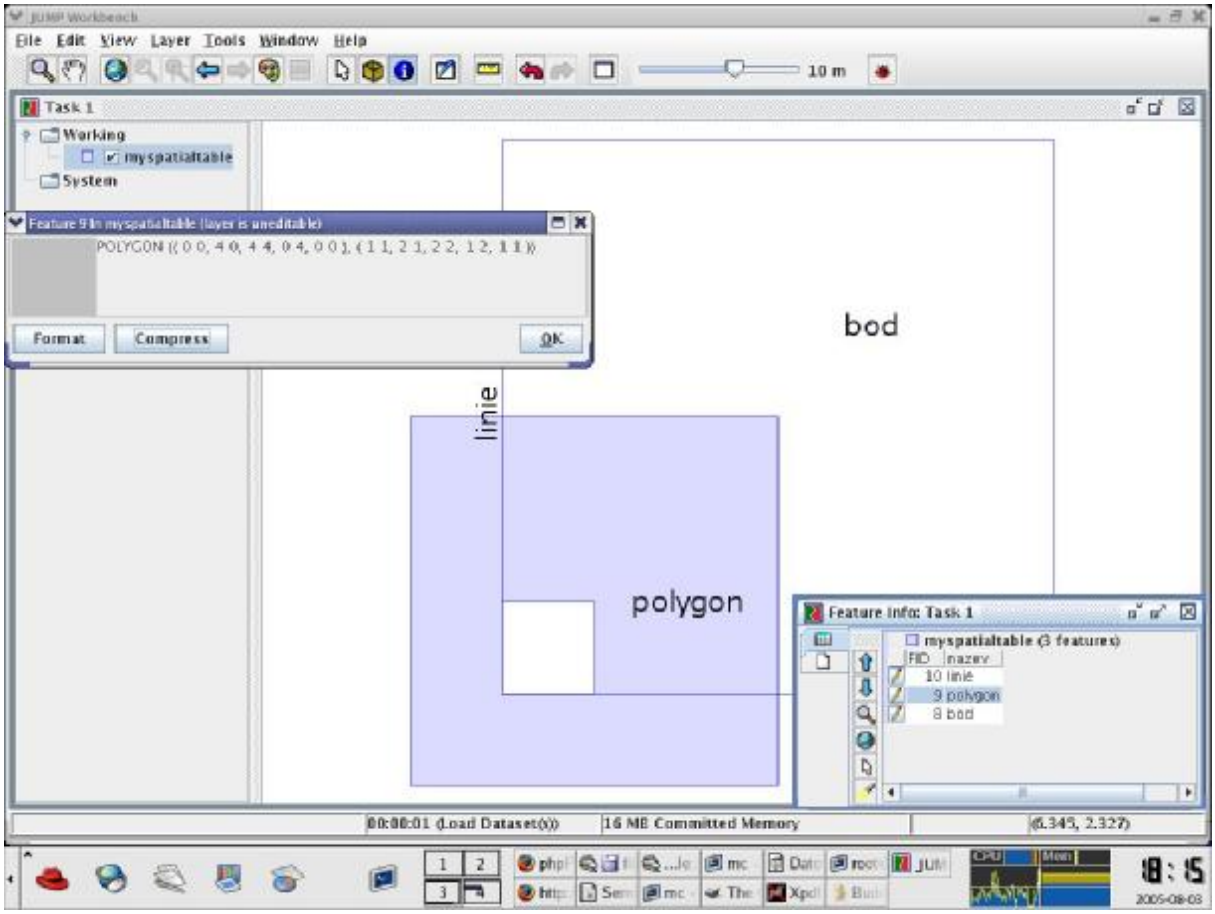
```
ogr2ogr -f PostgreSQL PG:'dbname=[nazev_databaze] user=[uzivatel]
password=[heslo]' [cesta_k_souboru.shp]
```

A dále ukázka pro export dat:

```
ogr2ogr -f "ESRI Shapefile" [tabulka] PG:'dbname=[nazev databaze]
user=[uzivatel] password=[heslo]'
```

Při exportu je ještě možné provádět export jen vybrané části dat specifikované např. oknem zadaným souřadnicemi nebo pomocí jednoduchého SQL příkazu:

```
ogr2ogr -f "ESRI Shapefile" [tabulka] PG:'dbname=[nazev databaze]
user=[uzivatel] password=[heslo]' -sql "SELECT [sloupce] FROM
[tabulka] WHERE [podminky]"
```



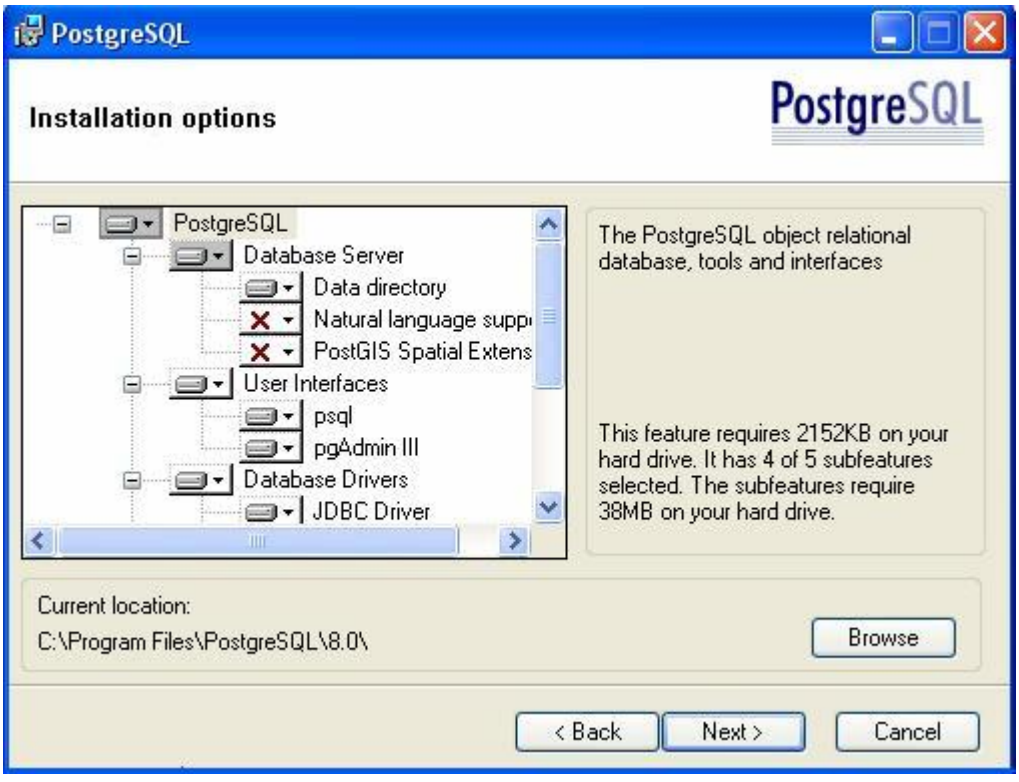
Obr. 2. Ukázka zobrazení vytvořených objektů v JUMP GIS.

6 INSTALACE PostgreSQL A PostGIS

V zásadě platí, že instalace programů je na OS Windows jednodušší ve srovnání s instalací na OS GNU/Linux, i když zde jsou také značné rozdíly dané používanou distribucí a její rozšířeností (dostupnost rpm a deb balíčků případně kompilace ze zdrojových kódů). Nicméně instalovat RDBMS PostgreSQL na OS Windows není pro větší a více využívané databáze vhodné, neboť výkon této databáze je ve srovnání s databází instalovanou na GNU/Linux výrazně nižší [3].

6.1 Instalace na OS Windows

Nejjednodušší cestou jak nainstalovat PostgreSQL na Windows je pomocí balíčku Windows Installer dostupného z adresy [4]. Pomocí něj nainstalujeme předkompilovanou verzi PostgreSQL společně s pgAdmin (nástroj pro grafickou administraci databáze), dále výběr z „contrib“ modulů poskytujících přídavné funkce (obr. 3) a můžeme vybrat i požadované procedurální jazyky.



Obr. 3. Ukázka dialogu při instalaci PostgreSQL na Windows.

PostgreSQL je možné instalovat jen na Windows 2000, XP nebo 2003. Instalátor při instalaci vytvoří účet služby, pokud tak bylo zadáno [6].

Pro instalaci extense PostGIS je potřeba ještě ze stránek [5] stáhnout a nainstalovat *dcmms*. Jeho instalace je jednoduchá a je podrobně popsána na stránkách *dcmms*.

6.2 Instalace na OS GNU/Linux

Dále uvádím postup instalace PostGIS 1.0.0, PostgreSQL 8.0.2 na GNU/Linux (Fedora Core 2), GNUMake 3.80, gcc version 3.3.3.

Instalace samotného PostgreSQL se dá provést buď kompilací ze zdrojového kódu nebo za použití rpm balíčků. Tato možnost je vhodná zejména pro případ dalších upgrade PostgreSQL. Je nutno ovšem dodat, že při reinstalaci PostgreSQL je potřeba opět provést kompilaci PostGISu. Je vhodné instalovat minimálně následující balíčky:

```
[root@kamik /]# rpm -qa | grep -i postgresql
postgresql-server-8.0.2-1PGDG
postgresql-8.0.2-1PGDG
postgresql-devel-8.0.2-1PGDG
postgresql-libs-8.0.2-1PGDG
```

Také je doporučeno instalovat knihovny Proj4 a GEOS (proj, proj-devel, geos, geos-devel a také gdal a gdal-devel). Proj4 umožňuje provádět transformace mezi různými zobrazeními a knihovna GEOS přidává možnosti práce s prostorovými daty (testování průniku, dotyků, určování rozdílů ploch, …). Zde je nutno ještě uvést, že v knihovně Proj4 je chybně definováno zobrazení S-JTSK.

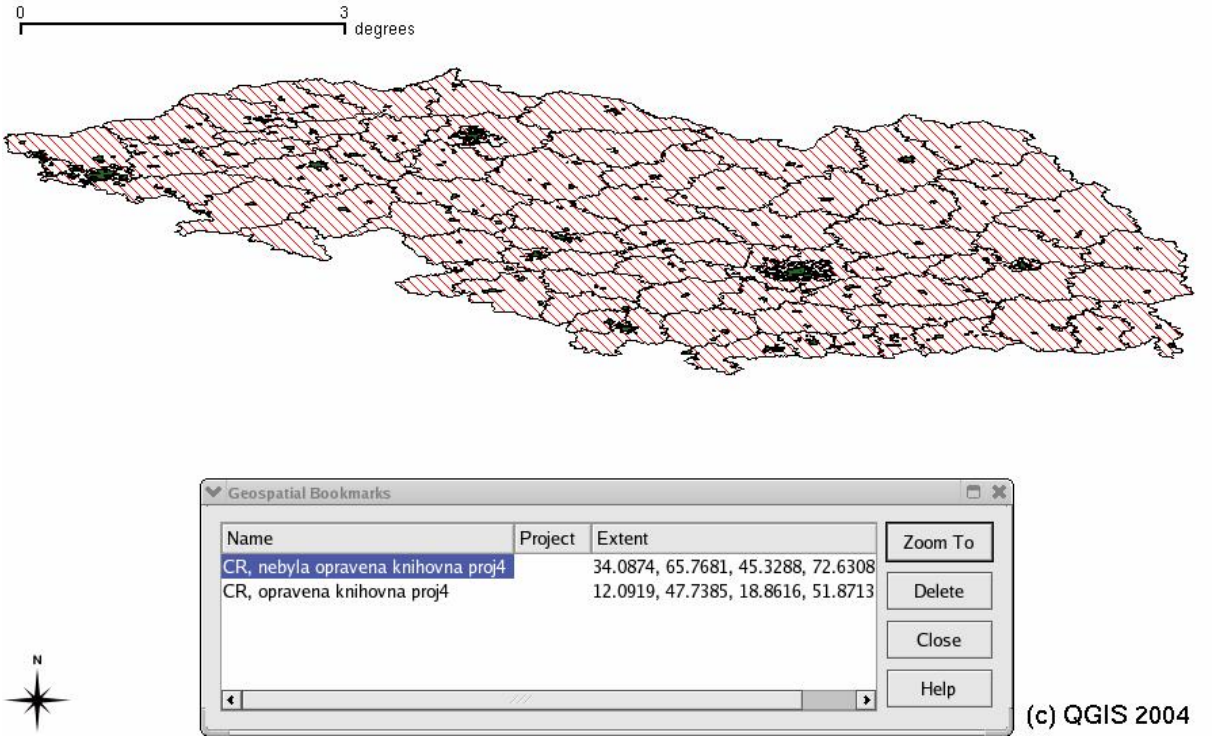
Dále je nutné z Internetu získat zdrojové kódy PostgreSQL a PostGISu (*http://www.postgresql.org/*, *http://postgis.refractions.net/*). PostgreSQL je potřeba rozbalit do vhodného adresáře a do podadresáře *contrib* dále rozbalit PostGIS. Před samotnou kompilací PostGISu je nutné v adresáři *postgis* editovat soubor *Makefile.conf*, kde je nutno změnit USE_GEOS=0 a USE_PROJ=0 na 1 a změnit příslušné defaultně nastavené cesty jen na */usr*. Poté v hlavním adresáři *postgresql* se spustí následující příkazy :

```
[root@kamik postgresql-8.0.2]#./configure
[root@kamik postgresql-8.0.2]#cd contrib/postgis-1.0.0
[root@kamik postgis-1.0.0]#make
[root@kamik postgis-1.0.0]#make install
```

Pokud nebyly hlášeny během kompilace a instalace žádné chyby, tak by tímto měla být instalace PostGISu hotova.

6.2.1 Oprava definice S-JTSK v knihovně proj4

V knihovně proj4, která je používaná při transformacích mezi jednotlivými souřadnicovými systémy, je chybně definováno zobrazení S-JTSK. Jsou zde sice definovány správně záporná znaménka, ale osy nebyly zaměněny [1]. Výsledky transformace z S-JTSK do WGS-84 bez opravy definice S-JTSK jsou zobrazeny na obr. 4 a dále na obr. 5 byla provedena transformace s použitím opravené knihovny proj4.



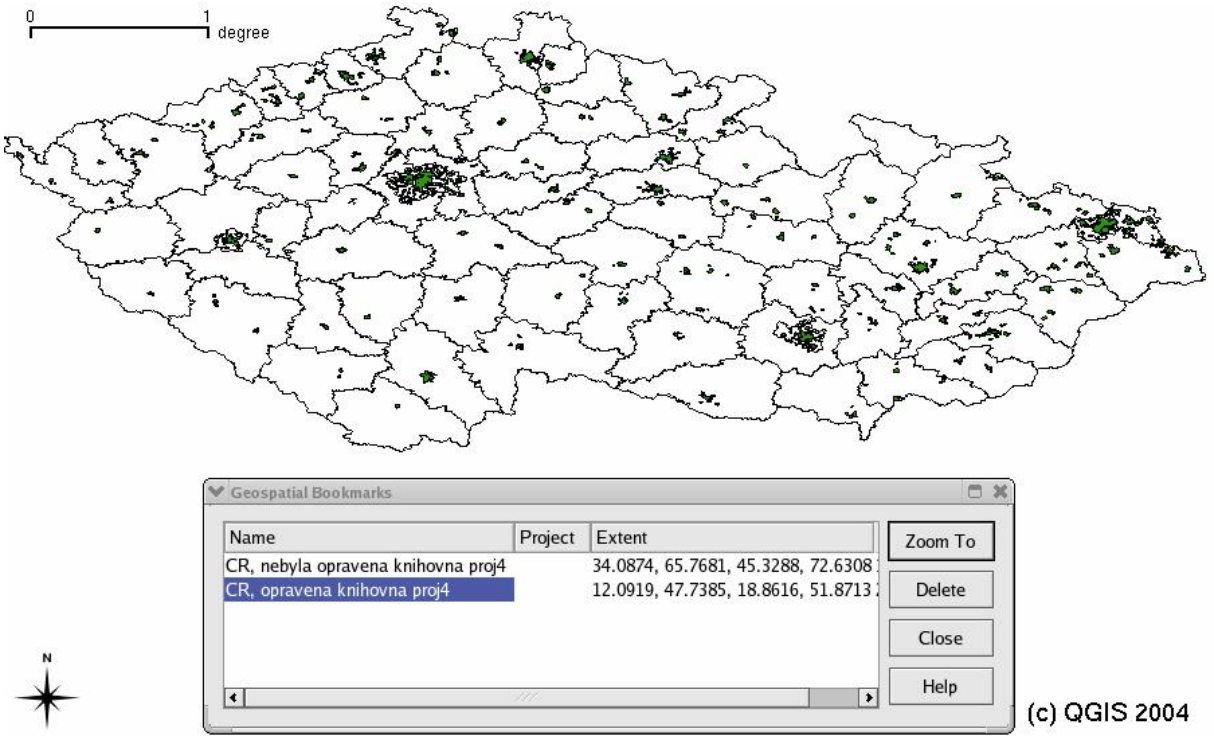
Obr. 4. Transformace S-JTSK do WGS-84 bez použité opravy knihovny proj4.

Pro opravu této chyby je nutno aplikovat na soubor PJ_krovak.c následující patch:

```
--- PJ_krovak.c.proj4   Sun Dec 15 23:31:04 2002
+++ PJ_krovak.c.upr     Mon May  5 10:24:42 2003
@@ -133,8 +133,8 @@
        ro = ro0 * pow(tan(s0 / 2. + s45) , n) / pow(tan(s / 2. +
s45) , n)   ;

    /* x and y are reverted! */
-       xy.y = ro * cos(eps) / a;
-       xy.x = ro * sin(eps) / a;
+       xy.y = -ro * cos(eps) / a;
+       xy.x = -ro * sin(eps) / a;

 #ifdef DEBUG
        strcpy(errmess,"a: ");
@@ -196,8 +196,8 @@
 /* Transformation */
```

```
    /* revert y, x*/
        xy0=xy.x;
-       xy.x=xy.y;
-       xy.y=xy0;
+       xy.x = -xy.y;
+       xy.y = -xy0;

        ro = sqrt(xy.x * xy.x + xy.y * xy.y);
        eps = atan2(xy.y, xy.x);
```

Pro případnou snadnou reinstalaci knihovny Proj4 je vhodné pomocí příkazu *rpmbuild* vytvořit z takto upravených zdrojových kódů rpm balíček.



Obr. 5. Transformace S-JTSK do WGS-84 s opravou knihovny proj4.

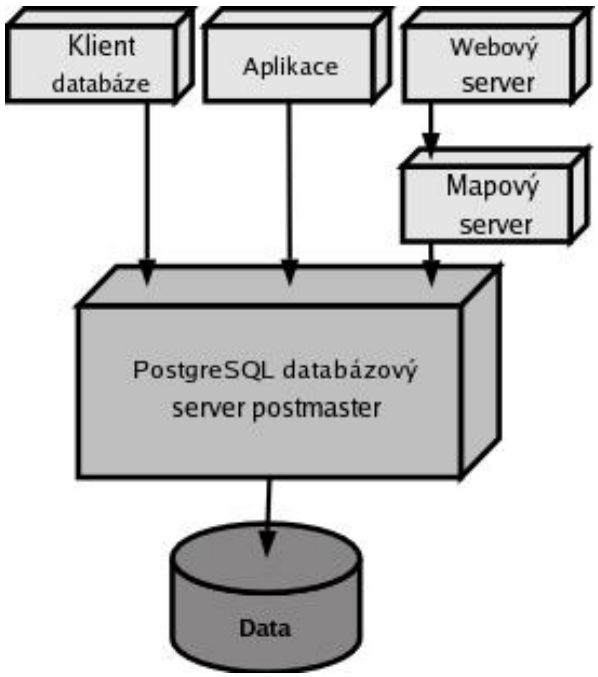
6.3 Zprovoznění a základní konfigurace serveru PostgreSQL

Základní konfigurace serveru PostgreSQL se provádí v souborech *postgresql.conf* a *pg_hba.conf*, které jsou umístěny v adresáři */var/lib/pgsql/data*. Zde je možné nastavovat např. maximální počet připojení k serveru, číslo portu, používání SSL, přidělenou paměť, přístupy uživatelů k databázím, … .

Přístupová práva k PostgreSQL se definují přímo na úrovni databáze pomocí souboru *pg_hba.conf*. K databázi je možné se připojovat několika různými klienty (obr. 6), přičemž každý uživatel bude muset zadat uživatelské jméno, heslo, databázi a tabulku, kterou chce zobrazit (obr. 7). Přístupová práva v souboru *pg_hba.conf* je možné nastavit např. takto:

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD

# "local" is for Unix domain socket connections only
local   all         postgres                          trust
local   all         all                               md5
# IPv4 local connections:
host    all         all         127.0.0.1/32          md5
host    all         all         0.0.0.0/0             md5
hostssl all         all         0.0.0.0/0             md5
# IPv6 local connections:
host    all         all         ::1/128               md5
```

Jak vidno z výpisu, je možné nastavovat povolení přístupu určitých uživatelů k daným databázím, a to jak lokálně, tak i vzdáleně. Přístup k databázím je nastaven tak, že jako uživatel postgres (správce databáze) se může lokálně přihlásit jen root. Metoda trust dovoluje přihlášení bez použití hesla. Všichni ostatní uživatelé musí zadávat hesla, a ta jsou pomocí metody *md5* přenášena v šifrované podobě.



Obr. 6. Schéma připojení klientů.

Při konfiguraci souboru *postgresql.conf* pro základní nastavení stačí povolit port=5432 a nastavit tcpip_socket jako true. Při následném pokusu o znovuspuštění služby postgresql dojde k selhání.

```
[root@kamik ~]# /etc/init.d/postgresql restart
Stopping httpd:                                            [  OK  ]
Starting httpd:                                            [FAILED]
```

Pro zjištění příčiny selhání startu postgresql je vhodné zkusit zprovoznit postmastr.

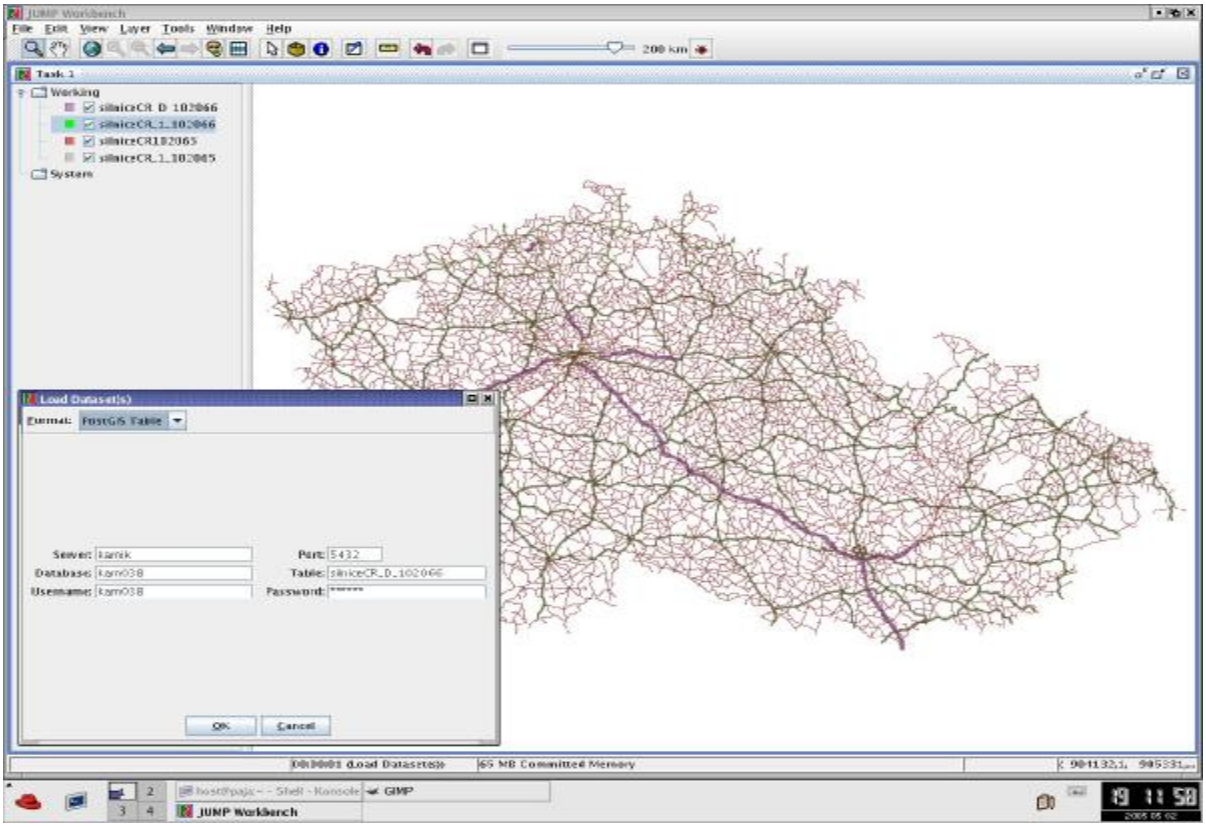
```
[root@kamik ~]# su postgres
bash-3.00$ /usr/bin/postmaster -D /var/lib/pgsql/data/
FATAL:  could not load server certificate file
"/var/lib/pgsql/data/server.crt": není souborem ani adresářem
```

Odstranění této chybové hlášky lze docílit zkopírováním certifikačních souborů, které se nachází v adresáři */etc/httpd/conf/ssl.crt/* podle níže uvedeného výpisu, nebo je také možné použít některou utilitu pro vytvoření vlastních certifikačních souborů, přičemž doporučuji první variantu.

```
[root@kamik ~]#cd /usr/share/ssl/certs/
[root@kamik certs]#make mujcertifikat.crt
```

```
cp /etc/httpd/conf/ssl.crt/server.crt /var/lib/pgsql/data/server.crt
cp /etc/httpd/conf/ssl.crt/server.crt /var/lib/pgsql/data/server.key
cat /etc/httpd/conf/ssl.key/server.key >>
/var/lib/pgsql/data/server.key
```

Zkopírovaným souborům je ještě nutné změnit vlastníka na postgres.



Obr. 7. Ukázka připojení k databázi pomocí JUMP GIS.

7 SPRÁVA DATABÁZE

Před prací s databází je nutné nejprve vytvořit uživatele, databázi a nastavit hesla uživatelům. Je zapotřebí se přihlásit jako uživatel *postgres*, jenž je správcem databáze PostgreSQL. Poté je nutné vytvořit uživatele *user* s parametrem -E, což znamená, že heslo uživatele má být šifrováno. Při vytváření databáze parametr -O udává, že vlastníkem databáze je uživatel *host*.

Dále musíme vytvořit v dané databázi procedurální jazyk plpgsql a následně do databáze importovat soubor *lwpostgis.sql* s SQL instrukcemi, které vytvoří nové funkce a datové typy. Poté ještě naimportujeme soubor *spatial_ref_sys.sql*, jenž obsahuje definice různých zobrazení. Pomocí interaktivního terminálu psql sloužícího ke správě databáze PostgreSQL můžeme nastavit heslo uživateli host. Celý postup tvorby databáze a uživatele uvádím níže.

```
[root@kamik ~]# su postgres
bash-3.00$ createuser -E host
bash-3.00$ createdb mojedb -O host
bash-3.00$ createlang plpgsql mojedb
bash-3.00$ psql -d mojedb -f /usr/local/pgsql/share/lwpostgis.sql
bash-3.00$ psql -d mojedb -f
/usr/local/pgsql/share/spatial_ref_sys.sql
bash-3.00$ psql -d mojedb
mojedb=# alter user host password 'mojeheslo';
```

7.1 Textový nástroj pro správu databáze

PostgreSQL obsahuje klienta v podobě textové konzole *psql* určeného pro správu databáze. *Psql* umožňuje interaktivně vkládat příkazy pomocí příkazové řádky a vracet výsledky zadaného dotazu. Případně je také možné vkládat příkazy pomocí textového souboru obsahujícího SQL příkazy (přepínač *–f filename*) nebo výstup přesměrovat do souboru pomocí přepínače *–o filename*.

Pokud se chceme připojit k databázi pomocí klienta *psql* je nutné znát minimálně jméno databáze ke které se chceme připojit. Dále bude od nás požadováno heslo vlastníka databáze nebo můžeme přidat jako argument jméno uživatele s přístupem k dané databázi. Příklad připojení k databázi pomocí klienta *psql* uvádím níže.

```
[root@kamik /]# ssh kam038@postgis.vsb.cz
Password:
Linux postgis.vsb.cz 2.6.13-ck6s #2 Tue Oct 4 09:24:37 CEST 2005
x86_64 GNU/Linux
Last login: Mon Feb 27 13:01:16 2006 from kola1308d.vsb.cz
kam038@postgis:~$ psql -d kam038
```

```
Password:
Welcome to psql 8.1.3, the PostgreSQL interactive terminal.
Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit
kam038=# \q
```

7.2 Grafické nástroje pro správu databáze PostgreSQL

Grafických nástrojů pro práci s databází PostgreSQL je celá řada (PgAccess, KPGsql, PGDesigner, …). Zde uvádím na ukázku jen dva nejrozšířenější klienty, a to sice webovou aplikaci phpPgAdmin a PgAdmin III.

7.2.1 phpPgAdmin

PhpPgAdmin je možné nalézt na Internetové stránce [7]. Pro jeho instalaci bude potřeba ještě HTTP server Apache, neboť se jedná o internetový nástroj. Apache je možné instalovat např. pomocí apt:

```
[root@kamik /]# apt-get install httpd
```

Po instalaci Apache se rozbalí phpPgAdmin-3.5.4.tar.bz2 do adresáře */var/www/html* a v adresáři */var/www/html/phpPgAdmin/conf/* spustíme následující příkaz:

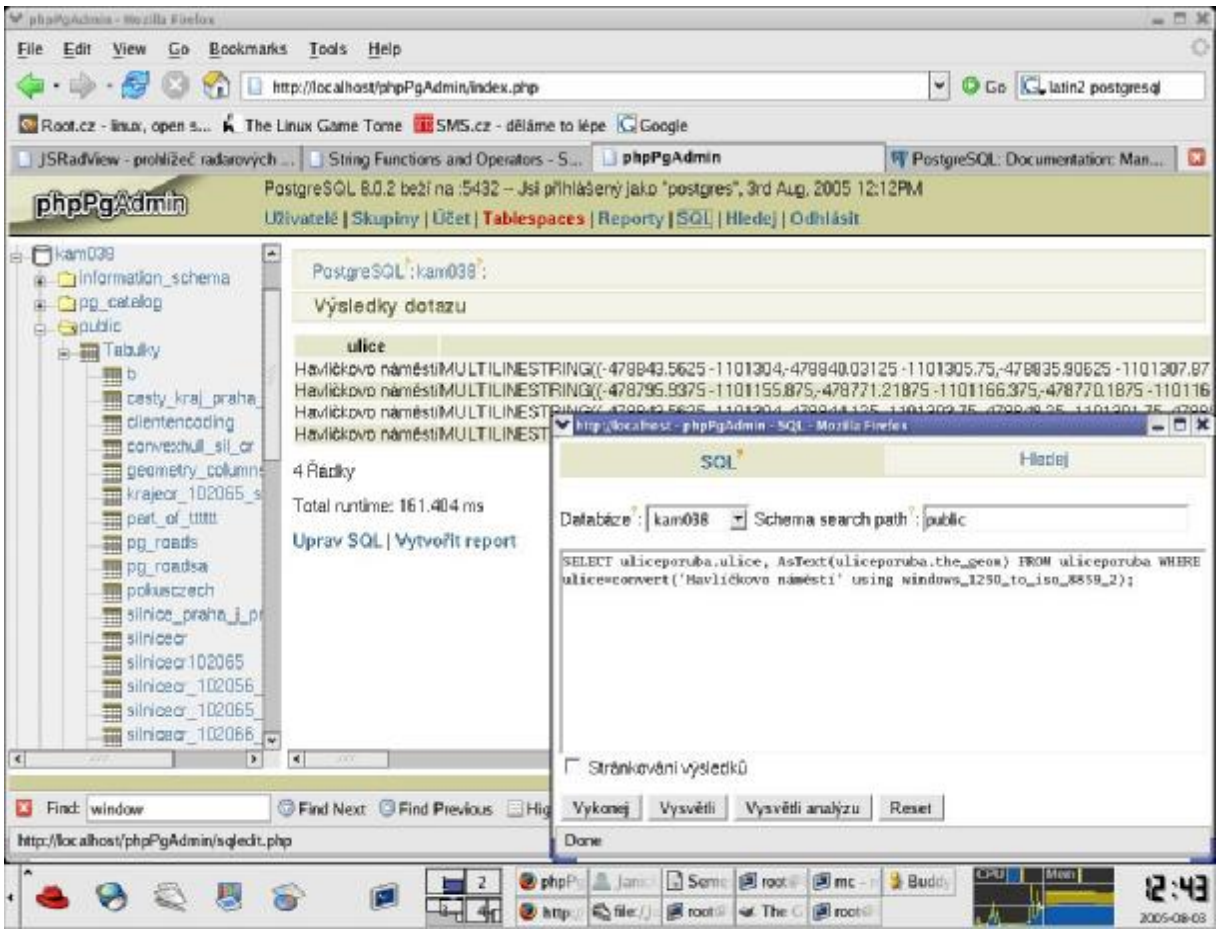
```
[root@kamik conf]#  cp config.inc.php-dist config.inc.php
```

Tímto byl vytvořen konfigurační soubor, ve kterém se dá nastavovat uživatelské prostředí a úroveň zabezpečení přístupu phpPgAdminu. Ke spuštění je potřeba ještě nastartovat http server Apache:

```
[root@kamik /]# /etc/init.d/httpd start
Starting httpd:                                            [  OK  ]
```

Pomocí prostřrdí phpPgAdmin je možné provádět veškerou správu databáze PostgreSQL, vytvářet dotazy, skupiny, uživatele, funkce, triggery, přidělovat práva, … .

Jeho nespornou výhodou je, že se jedná o webový administrativní nástroj, takže k jeho používání není potřeba na straně klienta instalovat žádný dodatečný software, stačí mít jen webový prohlížeč, který je součástí každé modernější instalace operačního systému. Na straně serveru je potřeba mít funkční webový server a kopii phpPgAdmin. Na obr. 8 je ukázka phpPgAdminu s formulářem pro SQL dotazy a zobrazenými výsledky.



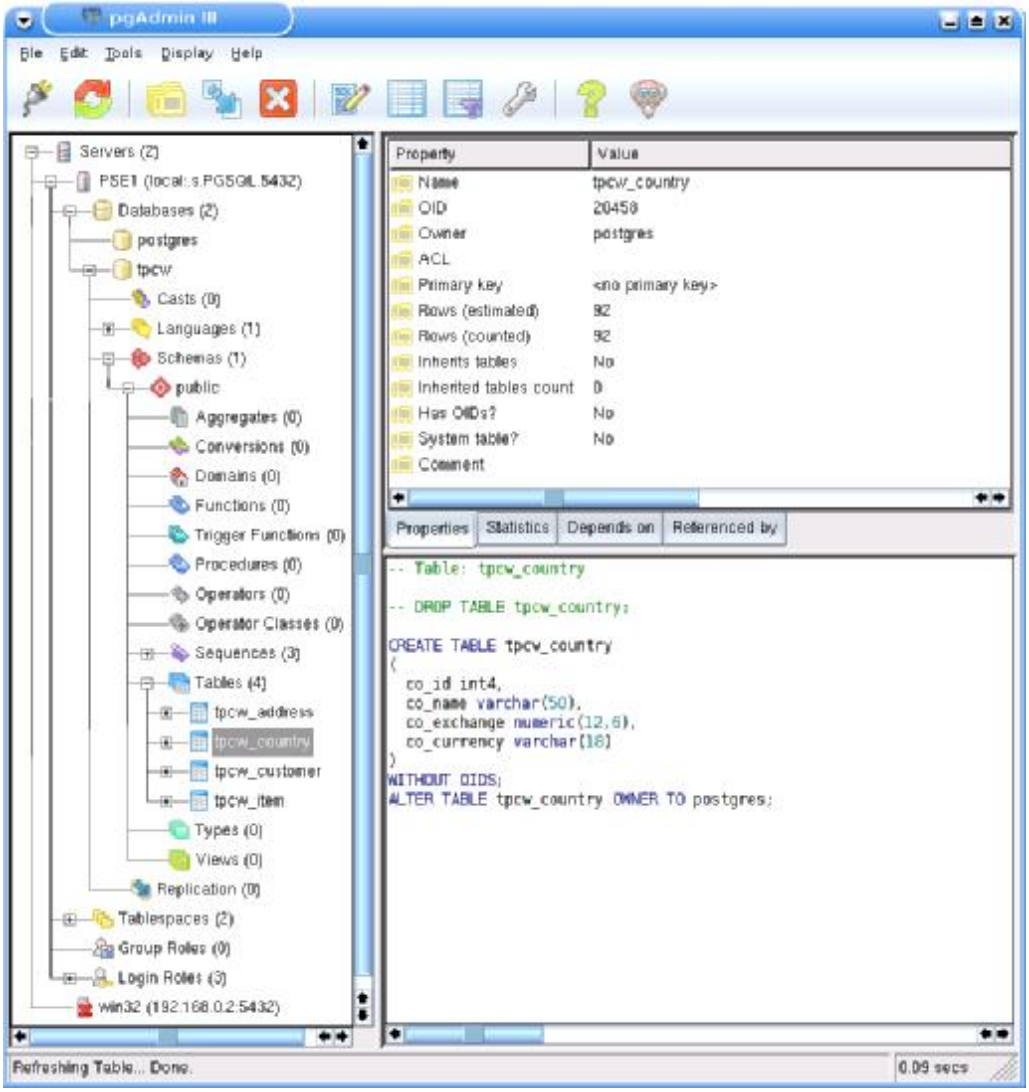
Obr. 8. Ukázka phpPgAdminu.

7.2.2 PgAdmin III

Jedná se o velmi rozšířený grafický nástroj určený pro administraci PostgreSQL. Je možné jej stáhnout ze stránky [8] nebo je distribuován společně s instalací PostgreSQL určenou pro Windows. Oproti phpPgAdminu je nutné tento software instalovat na každý počítač, neboť se nejedná o webovou aplikaci. Je k dispozici pro operační systémy Linux, FreeBSD, Solaris, Mac OSX a Windows.

Tento nástroj je určen pro široký rozsah uživatelů. Dovoluje provádět jednoduché dotazy nebo je také možné jej použít pro komplexní vývoj složitějšího projektu. Samozřejmostí je například zvýrazňování textu při tvorbě UDF a mnoho dalších funkcí.

Ukázka grafického vzhledu klienta PgAdmin III je na obr. 9.



Obr. 9. Ukázka PgAdmin III.

7.3 Další nástroje pro práci s PostgreSQL

Vzhledem k tomu, že SŘBD PostgreSQL je poměrně dlouho se rozvíjející databáze, existuje pro ni velké množství nástrojů. Pro ukázku jich několik uvádím níže [9]:

- **my2pg** - utilita v Perlu, která zkonvertuje dump databáze MySQL tak, že ho lze načíst do PostgreSQL;
- **pgdiff** - porovnává strukturu dvou PostgreSQL databází a vrací rozdíl jako sekvenci příkazů, které mohou být zadány do *psql* k přeměně struktury první tak, aby byla identická s druhou;
- **pg2xbase** - je soubor nástrojů pro konvertování databázových tabulek PostgreSQL z a do DBF databází;

- **Dia2Postgres** - je perlovský skript, který lze použít pro zkonvertování diagramů Dia do skriptů databáze PostgeSQL nebo do PHP minor classes;
- **auth postgresql** - umožňuje udržovat informace o uživatelích v databázi PostgreSQL místo v textových souborech (jak to je obvyklé), má výkonnější systém než NIS+;
- **Postgresql AutoDoc -** je nástroj, který dovoluje spouštění na PostgreSQL systému tabulek a vracet HTML, DOT, DIA a 2 styly XML, které popisují databázi;
- **Dbf2pg –** čte xBase soubory a pomocí SQL dotazů je zapisuje do tabulky v PostgreSQL.

8 ZÁKLADNÍ FUNKCE A SQL PŘÍKAZY PRO PRÁCI S DATABÁZÍ

SŘBD PostgreSQL obsahuje celou řadu funkcí, které je možné používat pro správu databáze. V této kapitole se budu zabývat jen těmi základními a těmi, které by mohli být užitečné při tvorbě datového skladu.

8.1 Nastavení přístupových práv k tabulkám

Nastavování přístupových práv se provádí pomocí příkazů GRANT a REVOKE, přičemž GRANT slouží k přidávání a REVOKE k odebírání oprávnění.

Oprávnění můžeme přidávat tabulkám a náhledům (view) jednomu nebo více uživatelům nebo skupinám. Nastavením práv implicitní skupině *public* takto získává práva každý uživatel v systému včetně těch, kteří budou později vytvořeni. Při vytvoření tabulky má veškerá práva jen její vlastník a jen vlastník může práva přidělovat. Vlastník může taktéž z bezpečnostních důvodů odebrat některá práva i sobě. Příklad udělení práv v terminálu *psql*:

```
kam038=# GRANT SELECT INSERT ON silnicecr TO paja;
kam038=# \z

Schema  |   Name   |  Type | Access privileges for database “kam038"
--------+----------+-------+---------------------------------------
public  |silnicecr |  view | {kam038=arwdRxt/kam038,paja=ar/kam038}
```

Ukázku použití příkazu REVOKE uvádím níže.

```
kam038=# REVOKE INSERT ON silnice FROM paja;
kam038=# \z

Schema |  Name   |   Type   |Access privileges for database "kam038"
-------+- -------+----------+---------------------------------------
public | silnice |   view   |{kam038=arwdRxt/kam038,paja=r/kam038}
```

Na výpisu je vidět, že uživatel *kam038* má veškerá práva a uživatel *paja* má jen právo výběru, které mu bylo uděleno uživatelem *kam038*, jenž je uveden za lomítkem. Uživatel, který nemá práva k dané tabulce/náhledu, se může pouze podívat na strukturu a datové typy, z nichž je tvořena.

Význam písmen vyjadřujících práva:

r - SELECT ("read")

w - UPDATE ("write")

a - INSERT ("append")

d - DELETE

R - RULE

x - REFERENCES

t - TRIGGER

arwdRxt -- ALL PRIVILEGES

8.2 Příkaz VACUUM

Při mazání nebo update záznamů v tabulkách nedochází k fyzickému odstranění záznamu z databáze, ale je jen označen za neplatný. Při použití příkazu *VACUUM* jsou všechny tyto přebytečné záznamy odstraněny. Spouštět příkaz *VACUUM* je doporučováno provádět periodicky případně vždy po větších updatech nebo mazání záznamů [10].

VACUUM se doporučuje spouštět společně s příkazem *ANALYZE*, který provede sběr statistických informací o tabulkách v dané databázi. Tyto informace jsou poté použity při tvorbě dotazovacího plánu k urychlení dotazu.

8.3 Tvorba náhledů (view)

Náhledy nejsou přímo „fyzicky“ uloženy na disku, ale jsou to jen určitým způsobem vybraná požadovaná data z tabulek a seskupená do jednoho objektu zvaného view. Při každém dotazu, kdy je dané view zmíněno, musí proběhnout dotaz, kterým byl náhled vytvořen. Náhledy jsou vhodné pro seskupování informací z více tabulek do jednoho objektu, nebo mohou být využity pro zobrazení určitých dat z tabulek, uživatelům, kteří k nim nemají přístup z bezpečnostních důvodů. Dále uvádím osnovu tvorby view a praktický příklad použití.

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW name [ ( column_name
[, ...] ) ] AS query
```

```
CREATE OR REPLACE VIEW my_tables AS SELECT
c.relname AS Name, r.rolname AS Owner FROM
pg_catalog.pg_class c
LEFT JOIN pg_catalog.pg_roles r ON r.oid = c.relowner
LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind IN ('r') AND pg_catalog.pg_table_is_visible(c.oid)
AND n.nspname !~ '^pg_' AND r.rolname IN (SELECT user);
```

Výše uvedený příklad vytvoří view *my_tables* obsahující názvy tabulek vlastněné přihlášeným uživatelem, který se na toto view dotáže.

8.4 Tvorba UDF

PostgreSQL umožňuje uživatelům tvořit své vlastní funkce i v jiných programovacích jazycích než jen v C nebo SQL. Tyto další jazyky jsou všeobecně nazývány procedural languages (PL). PostgreSQL nemá vestavěnou podporu pro interpretaci procedurálních jazyků, proto musí být zdrojový kód funkce předán handlenu určenému pro konkrétní jazyk. Handler je sdílená funkce napsaná v C a je při požadavku zavedena do paměti stejně jako jakékoliv další C funkce.

PostgreSQL obsahuje v současné době čtyři procedurální jazyky dostupné společně se standardní distribuci. Jsou to jazyky PL/pgSQL, PL/Tcl, PL/Perl a PL/Python. Mimo distribuci lze například použít PL/R, PL/PHP, PL/Java, … .

Pokud chceme používat např. procedurální jazyk plpgsql, je nutné jej nainstalovat do požadované databáze pomocí příkazu:

```
[root@kamik ~]# su postgres
bash-3.00$ createlang plpgsql nazev_databaze
```

V následující kapitole se pokusím jen lehce nastínit možnosti tvorby funkcí pomocí procedurálního jazyka PL/pgSQL.

8.4.1 Funkce v PL/pgSQL

Cílem této podkapitoly je ukázat základní schopnosti jazyka PL/pgSQL a rozhodně si neklade za cíl uvést konečný výčet vlastností.

Funkce psané v PL/pgSQL mohou přijímat nebo vracet jakékoliv datové typy, které jsou SŘBD PostgreSQL podporované, včetně vytvořených datových typů uživatelem. Také je možné psát funkce, které vrací typ proměnné RECORD, což je výsledek složený z datových typů, které jsou obsaženy v dotazu. Pokud není požadován žádný výsledek funkce, je možné definovat funkce vracející prázdný datový typ *void*.

Definice funkce musí být provedena v bloku s následujícím schématem:

```
[ <<label>> ]
[ DECLARE
    declarations ]
BEGIN
    statements
END [ label ];
```

Každý blok může být rozdělen na další logické podbloky vyznačené vždy pomocí BEGIN a END;.V těchto podblocích je možné provádět nové deklarace proměnných pomocí příkazu DEFINE.

PL/pgSQL obsahuje dva druhy komentářů, jeden začíná dvěmi pomlčkami (--) a je platný pro celý řádek a nebo je možné provést zakomentování celého bloku zdrojového kódu, který se ohraničí následujícími značkami stejně jako v C - /* a */.

Přístup k předaným parametrům při volání funkce je možné provést buď v sekci DEFINE pomocí ALIAS FOR:

```
...
DECLARE
    my_variable ALIAS FOR $1;
BEGIN
...
```

kde `$1` je pořadí předávané proměnné v argumentu volané funkce, nebo je možné se přímo ve funkci odkazovat na proměnnou zmíněným tvarem `$n` bez deklarace zástupné proměnné.

Stejně jako ostatní programovací jazyky obsahuje PL/pgSQL podmínky if, case, cykli while a for, kontrolu výjimek (raise exception), operátor pro spojování řetězců („||”), možnost deklarovat pole, textový výstup do terminálu *psql* (raise notice), … .

Pro spouštění SQL dotazů, jejichž parametry jsou předány jako argumenty funkci a jsou tedy proměnlivé, je nutné používat příkaz EXECUTE, který spouští předaný řetězec jako SQL dotaz. To má oproti staticky definovaným SQL dotazům (dotazům, u nichž se nemění parametry) tu nevýhodu, ze pokaždé musí být před vykonáním tohoto dotazu vytvořen prováděcí plán, což zpomaluje průběh funkce.

Tento příkaz můžeme použít i při iteraci přes záznamy dle příkladu uvedeného níže. Proměnná *pocitadlo* musí být datového typu RECORD - datový typ složený z typů daného řádku nebo ROWTYPE - datový typ určeného sloupce (např. DECLARE pocitadlo pg_catalog.pg_class%ROWTYPE).

```
...
FOR pocitadlo IN EXECUTE SQLvýraz LOOP
    příkazy
END LOOP
...
```

Funkce se vytváří pomocí příkazu:

```
CREATE FUNCTION nazev_funkce(datové_typy_parametrů) RETURNS typ AS$$
```

Níže jako ukázku uvádím funkci, která vytváří výřez z dané tabulky (parametr $1) obsahující prostorová data podle zadaných souřadnic (parametry $2-$5) a ukládá jej do stejnojmenné tabulky s příponou _cut.

```
CREATE OR REPLACE FUNCTION w_cut_by_window(text,real,real,real,real)
RETURNS void AS $$
--funkce vytvari vyrez z vybraneho okna pojmenovany *_cut podle
sourednic x1,y1,x2,y2, kde x1, y1 jsou sour. leveho dolniho rohu a
x2,y2 jsou souradnice praveho horniho rohu. $1 je nazev vrstvy
DECLARE
  srid1 int4;
  str text;
  str1 text;
  str2 text;
  str3 text;
  name_geom text;
  name_ text;

BEGIN
--kontrola zda existuje tabulka se stejnym nazvem, pokud ano, tak ji
smazu a pokracuju dal
  SELECT relname INTO name_ FROM pg_catalog.pg_class WHERE relkind
IN ('r') AND relname !~ '^pg_' AND
pg_catalog.pg_table_is_visible(pg_catalog.pg_class.oid) AND relname
= $1 ||'_cut';
  IF name_ IS NOT NULL THEN
    EXECUTE 'DROP TABLE '|| $1 || '_cut';
  END IF;
  SELECT srid INTO srid1 FROM geometry_columns WHERE f_table_name =
$1;
  SELECT f_geometry_column INTO name_geom FROM geometry_columns
WHERE f_table_name = $1;
str = 'CREATE TABLE ' || $1 || '_cut as SELECT intersection(r.' ||
name_geom ||',' || 'GeomFromText('||'\'POLYGON((' || $2 || ' ' || $3
|| ',' || $4 || ' ' || $3 || ',' || $4 || ' ' || $5 || ',' || $2 ||
' ' || $5 || ',' || $2 || ' ' || $3 || '))\','|| srid1 ||')' ||') AS
int_geom, r.* FROM ' || $1 || ' AS r WHERE r.' || name_geom || ' &&
' || 'GeomFromText('||'\'POLYGON((' || $2 || ' ' || $3 || ',' || $4
|| ' ' || $3 || ',' || $4 || ' ' || $5 || ',' || $2 || ' ' || $5 ||
',' || $2 || ' ' || $3 || '))\','|| srid1 ||')';
  EXECUTE str;
  str2 = 'ALTER TABLE ' || $1 || '_cut DROP COLUMN ' || name_geom ||
' CASCADE';
  EXECUTE str2;
  str3 = 'ALTER TABLE ' || $1 || '_cut RENAME COLUMN int_geom TO '
|| name_geom;
  EXECUTE str3;
END;
$$ LANGUAGE PLPGSQL
```

8.5 Tvorba spouštěčů (trigger)

Trigger je nástroj, který zajišťuje automatické provedení funkce zapsané pomocí jazyka SQL nebo jiného PL při vložení, zrušení nebo změně záznamu v určité tabulce. Trigger může například zajistit, že:

- před vložením nového záznamu do tabulky bude provedena kontrola jeho obsahu a doplněny hodnoty některých sloupců;
- po smazání záznamu se provedou následné akce;
- při změně hodnoty určitého sloupce v záznamu se porovná stará a nová hodnota a na základě jejich vztahu se provedou další akce.

Triggery zjednodušují tvorbu aplikací, protože přenášejí část práce databázové aplikace na server. Umožňují centralizované definování pravidel platných pro celý datový sklad.

Trigger je pojmenovaným objektem patřícím do databázové aplikace a má tyto vlastnosti:

- je svázán s určitou tabulkou a reaguje na změny v této tabulce;
- reaguje na právě jednu z SQL akcí INSERT, DELETE nebo UPDATE;
- provádí se buď před (BEFORE) provedením výše specifikované akce, nebo po ní (AFTER);
- pokud akce ovlivňuje více než jeden záznam, pak se může spouštět buď pro každý ovlivněný (tj. vložený, zrušený nebo změněný) záznam zvlášť, nebo pro všechny záznamy najednou;
- specifikuje akci (zapsanou v jazyce SQL), která bude automaticky provedena, jakmile jsou splněny podmínky pro spuštění triggeru.
- Při provedení konkrétní akce mohou být splněny podmínky pro spuštění více než jednoho triggeru. V takovém případě jsou spuštěny po řadě všechny.

Níže uvádím schéma tvorby triggeru, který provede asociaci požadované funkce s akcí prováděnou na tabulce.

```
CREATE TRIGGER nazev { BEFORE | AFTER } { udalost [ OR ... ] }
    ON tabulka [ FOR [ EACH ] { ROW | STATEMENT } ]
    EXECUTE PROCEDURE nazev_funkce ( argumenty )
```

Jako trigger můžeme použít libovolnou PL/pgSQL funkci bez parametrů vracející hodnotu typu *trigger*. Při provádění PL/pgSQL funkce jako triggeru máme k dispozici několik vestavěných proměnných z nichž uvádím jen některé. TG_OP popisuje příkaz vedoucí k spuštění triggeru, TG_NAME obsahuje název spuštěného triggeru, TG_WHEN obsahuje řetězec buď AFTER nebo BEFORE podle definice triggeru, v TG_RELID je uloženo *oid* tabulky, na které se spustil trigger, TG_RELNAME nese název tabulky převedený na málá písmena, na které se trigger spustil, NEW obsahuje celý řádek s novou verzí hodnot (pouze pro INSERT a UPDATE), OLD obsahuje řádek s původní verzí hodnot (pouze pro DELETE a UPDATE). Pokud funkce vrací NULL, tak se neprovede změna dat.

Jako malou ukázku tvorby triggerů uvádím níže trigger, který po vložení tabulky s prostorovými daty do databáze vloží do tabulky *basic_metadata* nový záznam obsahující název aktuelně vložené tabulky, jméno uživatele, který ji tam vložil a také je automaticky vytvořen časový záznam kdy byl trigger spuštěn. Tento čas je téměř identický s okamžikem vložení záznamu do tabulky *geometry_columns*. To je možné díky defaultně nastavenému datovému typu DEFAULT CURRENT_TIMESTAMP, který při vytvoření nového záznamu volá funkci *now()*.

Definice vytvořené tabulky je následující:

```
CREATE TABLE basic_metadata(name text, owner name, time timestamp
DEFAULT CURRENT_TIMESTAMP) ;
```

Funkce, která bude volána jako trigger nesmí přebírat žádné parametry a musí vracet typ trigger. V proměnné NEW je uložen celý nový řádek, který je vkládán do tabulky *geometry_columns*. Přístup k jednotlivým prvkům záznamu je prováděn pomocí známé tečkové notace.

```
CREATE OR REPLACE FUNCTION insert_new() RETURNS trigger AS $$
DECLARE
  str text;
  u name;

BEGIN
  SELECT INTO u user;
  str := 'INSERT INTO basic_metadata(name, owner) VALUES
(\''||NEW.f_table_name||'\',\''||u||'\');';
  RAISE NOTICE '%', str;
  EXECUTE str;
  RETURN NULL;
END;
$$LANGUAGE PLPGSQL
```

Následující výraz vytváří spoušť, která je aktivována pro každý nový vložený řádek poté, co je do tabulky *geometry_columns* vložen nový záznam. V tomto případě vždy jen jednou, neboť je možné do databáze vkládat pouze jednu tabulku a tedy provádět jen jeden zápis do tabulky *geometry_columns.*

```
CREATE TRIGGER ins_mtdta AFTER INSERT ON geometry_columns FOR EACH
ROW EXECUTE PROCEDURE insert_new();
```

Funkce v PL/pgSQL a tedy i triggery jsou spouštěny jako klasické transakce, což znamená, že v případě selhání jakéhokoliv příkazu dojde k navrácení veškerých změn, které byly doposud vykonány. Z těchto důvodů, je-li žádoucí i přes selhání pokračovat ve vykonávání funkce, je možné v těle funkce odchytávat výjimky pomocí EXCEPTION dle následujícího schématu popisujícího ošetření výjimky dělení nulou:

```
...
EXCEPTION
  WHEN division_by_zero THEN
    RAISE NOTICE 'caught division_by_zero';
...
```

Seznam dalších výjimek lze nalézt na adrese [11].

8.6 Indexy

PostgreSQL server nabízí 4 typy indexů - B-tree (tento je defaultní), R-tree, GiST a Hash. Lze indexovat nejen jeden sloupec, ale i několik do jednoho indexu, je-li dle této kombinace pravidelně vyhledáváno, případně indexovat za použití funkce.

Základní vlastnosti jednotlivých typů indexů:

- **B-tree** - základní typ indexu, který je vytvářen automaticky při neuvedení typu indexu. Pro server je tato indexace nejvíce urychlující operace <, <=, =, >=, >, LIKE, ILIKE, ~ a ~*;
- **R-tree** - typ indexu optimalizovaný pro geometrická data. Pro operátory <<, &<, &>, >>, @, ~= a &&;
- **Hash** - index, je nejpomalejší na vytvoření. Rychlý pro porovnávání řetězců (respektive jejich hashů), funguje pouze pro operátor =;
- **GiST** (zobecněný vyhledávací strom) - Jedná se o rozšiřitelnou strukturu, která sdružuje mezi jinými vlastnosti B-trees a R-trees.

Indexů lze pro každou tabulku vytvořit libovolné množství, ale při jejich neuváženém použití může naopak dojít k citelnému zpomalení práce s databází, zejména při vkládání.

Kromě neomezeného počtu indexů pro tabulku lze indexy definovat i přes několik sloupců tabulky. Logicky, pokud se omezují řádky ve výběru tabulky (podmínkami v části WHERE) a děje se tak pravidelně v určité kombinaci sloupců, je vhodné tuto kombinaci sloupců zaindexovat. Index je pak uspořádán, že je první zaindexován první sloupec, poté druhý, třetí, ... Maximální počet sloupců pro zaindexování v jednom indexu je dle dokumentace 32 [13]. Multisloupcové indexy lze definovat jen pro B-tree a GiST.

Indexy ať už na jednom sloupci, nebo multisloupcové mohou mít vynucenu i informaci, že musí být unikátní. V současné době je tato vlastnost podporována pouze u B-tree indexů.

Indexace pomocí funkce má význam, pokud se k datům ve sloupci přistupuje pravidelně s použitím funkce.

8.6.1 Vytváření a rušení indexů

Pro vytvoření indexů slouží příkazy:

```
CREATE [UNIQUE] INDEX jmeno ON tabulka
[USING typ_indexu] (sloupec [trida_operatoru]
[, sloupec [trida_operatoru], ...])[ WHERE podminky];
CREATE [UNIQUE] INDEX jmeno ON tabulka
[USING typ_indexu] ( funkce (sloupec[,...]) [trida_operatoru]
[,...]);
CREATE [UNIQUE] INDEX jmeno ON tabulka
[USING typ_indexu] (sloupec [trida_operatoru]
[, sloupec [trida_operatoru], ...]);
```

Trida_operatoru určuje operátory, které může server v indexu pro daný sloupec použít, jejich příklad lze nalézt v originální dokumentaci.

Část **WHERE** slouží k vytvoření tzv. částečných indexů. Jedná se o kompromisní volbu mezi velikostí databázových souborů a rychlostí. Například mohou být v databázi taková data, kdy se v 98 % případů přistupuje na 10 % řádek, z tohoto důvodu má význam indexovat těch 10 % řádek a zbytek nechat neindexován s tím, že pokud uživatel potřebuje tato málo používaná data, musí déle čekat.

Indexy se ruší příkazem:

```
DROP INDEX jmeno [, dalsi_jmeno [, ...]] [CASCADE | RESTRICT];
```

Ve většině případů stačí jednoduché DROP INDEX jmeno_indexu, přídavné příkazy CASCADE (zrušení všech objektů závislých na rušeném indexu) a RESTRICT (zákaz rušení objektů závislých na indexu) najdou použití ve složitějších databázích.

Občas se může vyskytnout potřeba vynutit opravu indexů na databázi, tato nutnost vzniká ve dvou případech. Bud byl index poškozen a obsahuje neplatná data (tato varianta by neměla nikdy nastat), nebo při velkém množství indexů se mohou vyskytnout místa, kde nebyl index regenerován. K vynucení opravy se používá příkaz:

```
REINDEX {DATABASE | TABLE | INDEX} jmeno [FORCE]
```

Pro reindexaci všech indexů v databázi se použije REINDEX DATABASE ..., pro tabulku REINDEX TABLE ..., a pro jeden index REINDEX INDEX ... Parametr FORCE nemá význam, je jen kvůli kompatibilitě se staršími verzemi a jeho význam byl, že reindexace se provedla vždy i když indexy byly v pořádku.

Pokud bude prováděn import velkého množství dat do tabulky, která má již vytvořeny indexy, je doporučeno před importem indexy zrušit, provést vložení dat a poté opět indexy znovu vytvořit. Tento postup je volen z toho důvodu, že při vkládání velkého množství řádků dochází k enormnímu vytížení databáze způsobeného neustálou reindexací a může dojít k přerušení vkládání dat [12].

Při vkládání 1.000.000 řádků do tabulky o osmi sloupcích typu *real* bez vytvořených indexů byl zaznamenán čas 26.523,834 ms zatímco při vytvořeném B-tree indexu pouze na jednom sloupci na stejné tabulce byl zaznamenán čas 619.144,666 ms. Tyto výsledky jsou pouze informativního charakteru, neboť měření nebyla prováděna opakovaně, ale přesto je z nich patrné, že rozdíly ve vytíženosti SŘBD jsou značné.

8.6.2 Vliv B-tree indexů na rychlost vyhledávání v databázi

Aby bylo možno porovnat vliv B-tree indexace na rychlost vyhledávání v databázi, byla vytvořena funkce, která vytvoří tabulku s třemi sloupci datového typu *real* a naplní ji velkým množstvím (1.000.000) náhodných hodnot. Funkci uvádím níže:

```
CREATE OR REPLACE FUNCTION rand() RETURNS void AS $$
DECLARE
i int4;

BEGIN
  CREATE TABLE test_idx(col1 real, col2 real, col3 real);
  FOR i IN 1..1000000 LOOP
    INSERT INTO test_idx(col1, col2, col3) VALUES (random(),
```

```
random(), random());
  END LOOP;
END;
$$ LANGUAGE 'PLPGSQL'
```

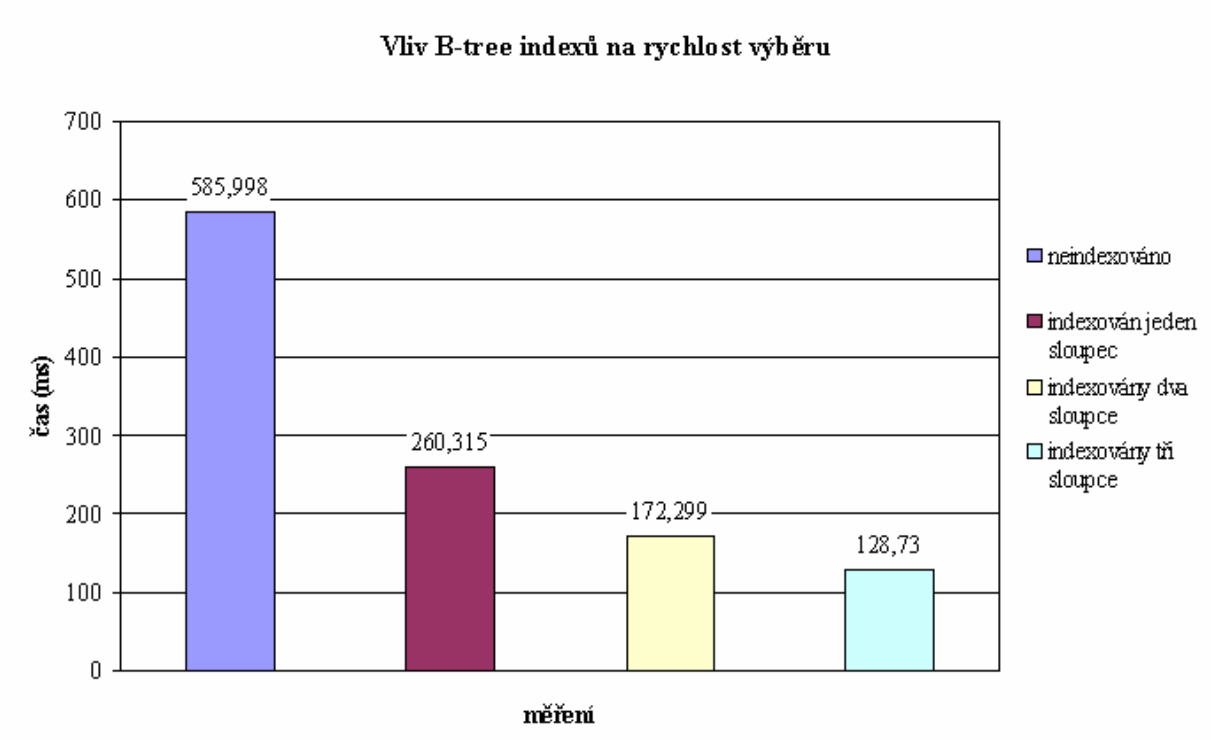
Poté byla provedena série dvaceti následujících dotazů na neindexované tabulce a výsledné časy byly zprůměrovány.

```
SELECT count(*) AS pocet FROM test_idx WHERE
(col1 > 0.5 AND col1 < 0.7) AND (col2 > 0.5 AND col2 < 0.7) AND
(col3 > 0.5 AND col3 < 0.7);
```

Dále byl vytvořen B-tree index pouze na prvním sloupci:

```
CREATE INDEX index_test_idx ON test_idx (col1);
```

Opět byla zopakována série dotazů, byly zprůměrovány a zrušeny vytvořené indexy a vytvořeny nové pro dva sloupce (*col1, col2*). Dále se celý postup opakuje až nakonec je vytvořen index pro všechny tři sloupce. Do výpočtu nebyl zařazován čas prvního provádění dotazu, neboť planner teprve vytváří prováděcí plán a tudíž se tento čas významně liší od ostatních. Konečné výsledky jsou zobrazeny na obr. 10



Obr. 10. Vliv B-tree indexace na rychlost výběru při použití operátoru „>”.

Z obrázku je zřetelné, že už jen při indexaci jednoho sloupce dochází k polovičnímu zkrácení času potřebného pro provedení dotazu. Indexace dalších sloupců má také poměrně významný vliv ačkoliv změna již není tolik markantní.

8.6.3 Vliv Hash indexů na rychlost vyhledávání v databázi

Významným omezením HASH indexů oproti B-tree je skutečnost, že je možné indexovat pouze jeden sloupec. Porovnání rychlosti HASH indexů bylo prováděno na tabulce obsahující téměř 409.000 záznamů. Nejprve byly prováděny níže uvedené dotazy **bez** indexace s výslednou průměrnou hodnotou času **318,995 ms**.

```
SELECT count(*) FROM europe_roads_01 WHERE cntryname = 'Portugal'
AND natlcode1 = 'A1'
```

Dále byl vytvořen HASH index pro sloupec *cntryname*:

```
CREATE INDEX index_europe_roads01_hash ON europe_roads_01 USING HASH
(cntryname)
```

S použitím **HASH** indexu bylo dosaženo při stejném dotazu průměrného času **5,896 ms**. Pro srovnání s B-tree indexy byl ještě vytvořen **B-tree** index na sloupec *cntryname* a bylo dosaženo průměrného času **5,748 ms** a dále byl vytvořen B-tree index na **oba sloupce** (*cntryname,natlcode1*). Celkový průměrný výsledek činil **2,966 ms**.

Srovnáním výsledků průměrných časů při indexaci pomocí HASH, B-tree a bez indexace je zřejmé, že HASH indexy jsou téměř stejně rychlé jako B-tree, ale není možné indexovat více sloupců zároveň, což je nepříjemné omezení ve srovnání s B-tree. Dále při indexaci a reindexaci dat pomocí HASH dochází k mnohem většímu zatížení SŘBD, což by při vkládání velkého množství dat mohlo mít za následek nechtěné přerušení transakce.

8.6.4 Vliv GiST indexů na rychlost prostorových dotazů

GiST indexy se používají ke zrychlení vyhledávání všech nepravidelných datových struktur jako jsou například záznamy složené z datového typu *array* apod., které není možné indexovat pomocí B-tree indexů. Syntaxe tvorby GiST indexu je následující:

```
CREATE INDEX [nazev_indexu] ON [tabulka] USING GIST
[geometricky_sloupec]
```

Po vytvoření indexu je doporučováno spustit příkaz VACUUM ANALYZE, která vytvoří statistiky o tabulkách, které budou následně použity při tvorbě optimálního prováděcího plánu.

GiST indexy mají oproti R-tree indexům v PostgreSQL dvě hlavní výhody. Zaprvé, GiST indexy jsou „NULL safe“, což znamená, že je pomocí nich možné indexovat i sloupce, které obsahují hodnoty NULL a zadruhé, GiST indexy podporují koncept tzv. lossiness, což je důležité pokud se pracuje s velmi velkými GIS objekty. Podpora „lossiness“ umožňuje

indexovat pouze „významnou“ část daného objektu, který je v případě GIS objektů jejich obálka. V případě R-tree indexů by došlo k chybě při tvorbě.

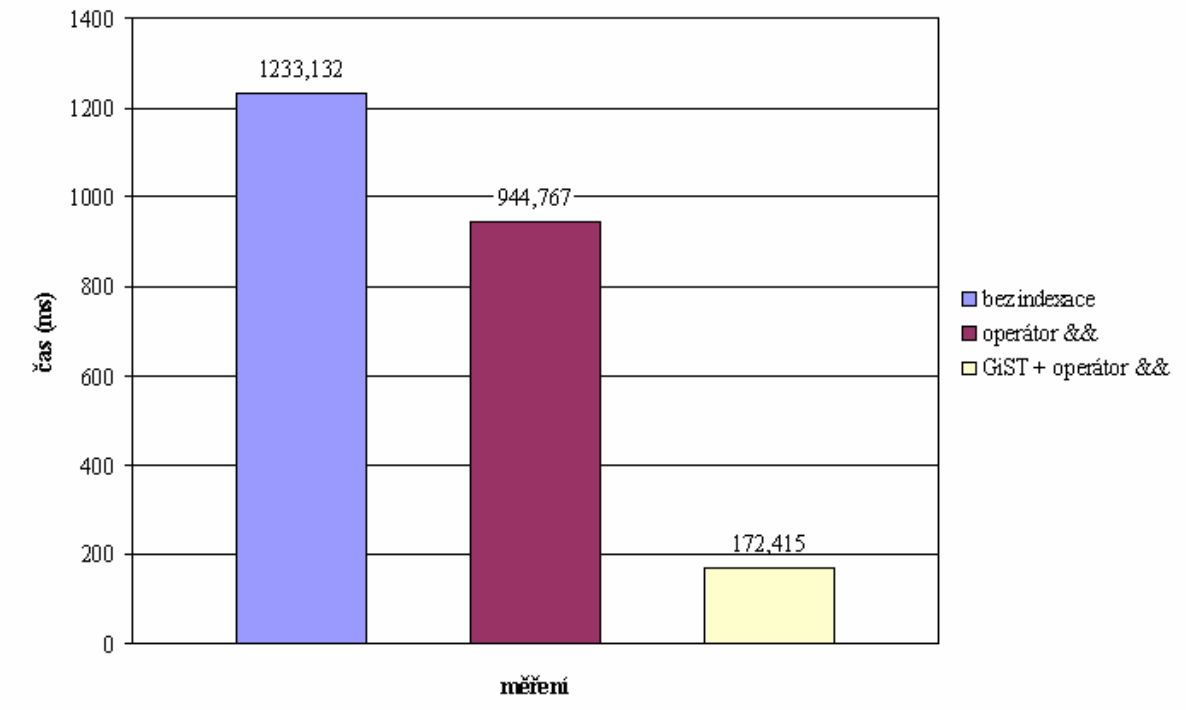
Pokud chceme používat při prostorových dotazech vytvořené GiST indexy, je důležité v nich používat operátory založené na využití hraničního rámce, jako je například operátor &&. Ostatní funkce, nejsou schopny využívat GiST indexy. Použitím operátoru && vymezíme oblast dotazu na určitý rámec, a ostatní funkce, které nejsou schopny využívat vytvořeného indexu budou pracovat jen v námi vymezené oblasti, čímž se velmi výrazně zvýší rychlost dotazu.

```
Pro testování rychlosti byla použita silniční síť Evropy o celkovém
počtu 408983 záznamů. Testování bylo prováděno dvěma způsoby dotazů,
z nichž první prováděl dotaz jen na geometrické objekty a druhý
kombinoval dotaz na geometrické a popisné atributy dat.
```

Dotaz zaměřený na geometrickou složku dat

Nad tabulkou byl prováděn dotaz uvedený níže, který vybírá elementy silniční sítě vzdálené maximálně 200 km od Prahy. Měření času byla prováděna 20x a výsledný čas byl zprůměrován. Po vytvoření indexů byl vždy spouštěn příkaz VACUUM ANALYZE.

```
CREATE TABLE x AS SELECT * FROM europe_roads_01 WHERE (distance
(the_geom_sjtsk, GeomFromText( 'POINT(-743380.9 -1043676.7)',
102065)) < 200000);
```



Obr. 11. Porovnání výkonu při použité GiST indexaci a použití operátoru &&.

A stejný dotaz využívající operátor && pro vymezení zájmové oblasti:

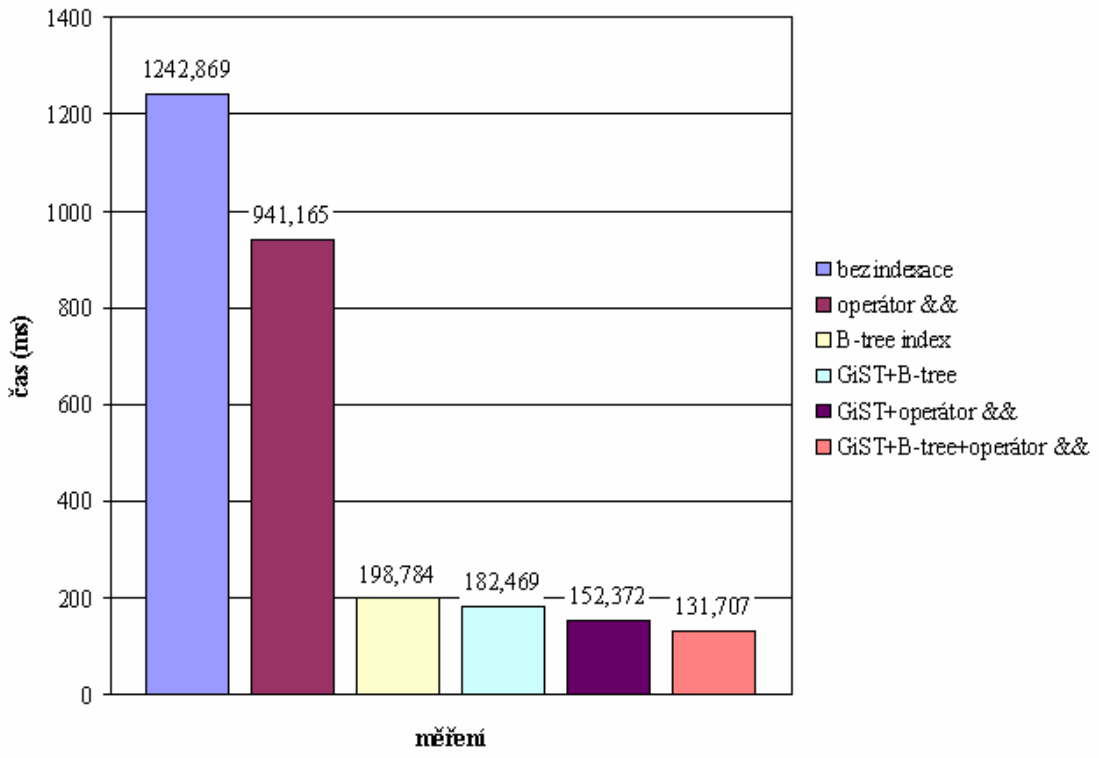
```
CREATE TABLE x as SELECT * FROM europe_roads_01 WHERE
(the_geom_sjtsk && GeomFromText('POLYGON((-943380.9 -843676.7, -
543380.9 -843676.7, -543380.9 -1243676.7, -943380.9 -1243676.7, -
943380.9 -843676.7))',102065)) AND (distance( the_geom_sjtsk,
GeomFromText( 'POINT(-743380.9 -1043676.7)', 102065 )) < 200000);
```

Výsledky jsou zobrazeny na obr. 11, z něhož je patrná velká úspora času při použití GiST indexů a operátoru &&. Dále je patrné, že při nevytvořených GiST indexech, ale za použití operátoru && došlo také k menšímu zvýšení výkonu (prostřední sloupec).

Kombinace dotazu na geometrické a popisné atributy dat

Dále bylo provedeno měření výkonu SŘBD při použití kombinovaných dotazů na geometrické a popisné atributy dat za použití GiST a B-tree indexů. Dotaz stejně jako v předešlém případě vyhledává segmenty silniční sítě vzdálené maximálně 200 km od Prahy a navíc přidává omezení, že tyto segmenty musí být v České Republice nebo v Polsku. Byl užit následující dotaz:

```
CREATE TABLE x as SELECT * FROM europe_roads_01 WHERE (distance(
the_geom_sjtsk, GeomFromText( 'POINT(-743380.9 -1043676.7)', 102065
)) < 200000) AND (cntryname='Czech Republic' OR cntryname='Poland')
```



Obr. 12. Kombinované dotazy na geometrické a popisné atributy při použití GiST a B-tree indexů.

a dále s operátorem *&&*:

```
CREATE TABLE x as SELECT * FROM europe_roads_01 WHERE
(the_geom_sjtsk && GeomFromText('POLYGON((-943380.9 -843676.7, -
543380.9 -843676.7, -543380.9 -1243676.7, -943380.9 -1243676.7, -
943380.9 -843676.7))',102065)) AND (distance(the_geom_sjtsk,
GeomFromText( 'POINT(-743380.9 -1043676.7)', 102065 )) < 200000) AND
(cntryname='Czech Republic' OR cntryname='Poland')
```

Byli vytvořeny různé kombinace dotazů současně s GiST a B-tree indexy, nebo bez některého z nich, případně oba chyběli. Také byl zkoumán vliv použití *&&* operátoru. Výsledky jsou zobrazeny na obrázku 12.

Dle očekávání bylo dosaženo nejlepších výsledků při kombinaci operátoru && a použití GiST a B-tree indexů. Z obrázku je také patrné, že opět dochází shodně jako v předchozím měření k mírnému zlepšení výkonu už jen při použití operátoru && bez použití jakékoliv indexace.

Z provedených měření je zřejmé, že při vytvoření vhodných indexů a jejich správném používání současně s operátory pracujícími s hraniční oblastí, včetně přiměřeného používání příkazu VACUUM ANALYZE, dojde k nezanedbatelnému zvýšení výkonu databáze i při složitých prostorových dotazech.

8.7 Konverze znakového kódování

Většina dat, která jsou k dispozici a obsahují české popisné atributy, byla vytvořena v prostředí Windows a tudíž používají kódování win1250. Pokud jsou uvedená data již importována do databáze, může to někdy způsobovat problémy při dotazech nebo při zobrazování popisků na mapě v prostředí Windows nebo GNU/Linux. Řešením je buď celý sloupec překonvertovat do požadovaného kódování, nebo vytvořit další sloupec, obsahující stejné atributy v požadovaném kódování. K tomuto se používá funkce *Convert()*. Způsob použití funkce *Convert()* uvádím dále:

```
UPDATE okresy SET nazev_utf8=Convert(nazev, ´WIN1250´,´UTF8´);
```

Výše popsaný SQL příkaz vloží do sloupce *nazev_utf8* překonvertované atributy ze sloupce *nazev* v cílovém kódování UTF-8 ze znakového kódování win1250. Seznam možných kódování lze nalézt na adrese [17].

Z uvedených důvodů vyplývá, že je vhodné importovat data do databáze přímo v požadovaném kódování databáze. Za tímto účelem obsahuje utilita shp2pgsql, která se

používá k importu souborů ESRI Shapefile do databáze, přepínač –W následovaný názvem znakového kódování zdrojového souboru.

Dále uvádím příklad použití specifikace znakového kódování LATIN2 přímo při importu dat:

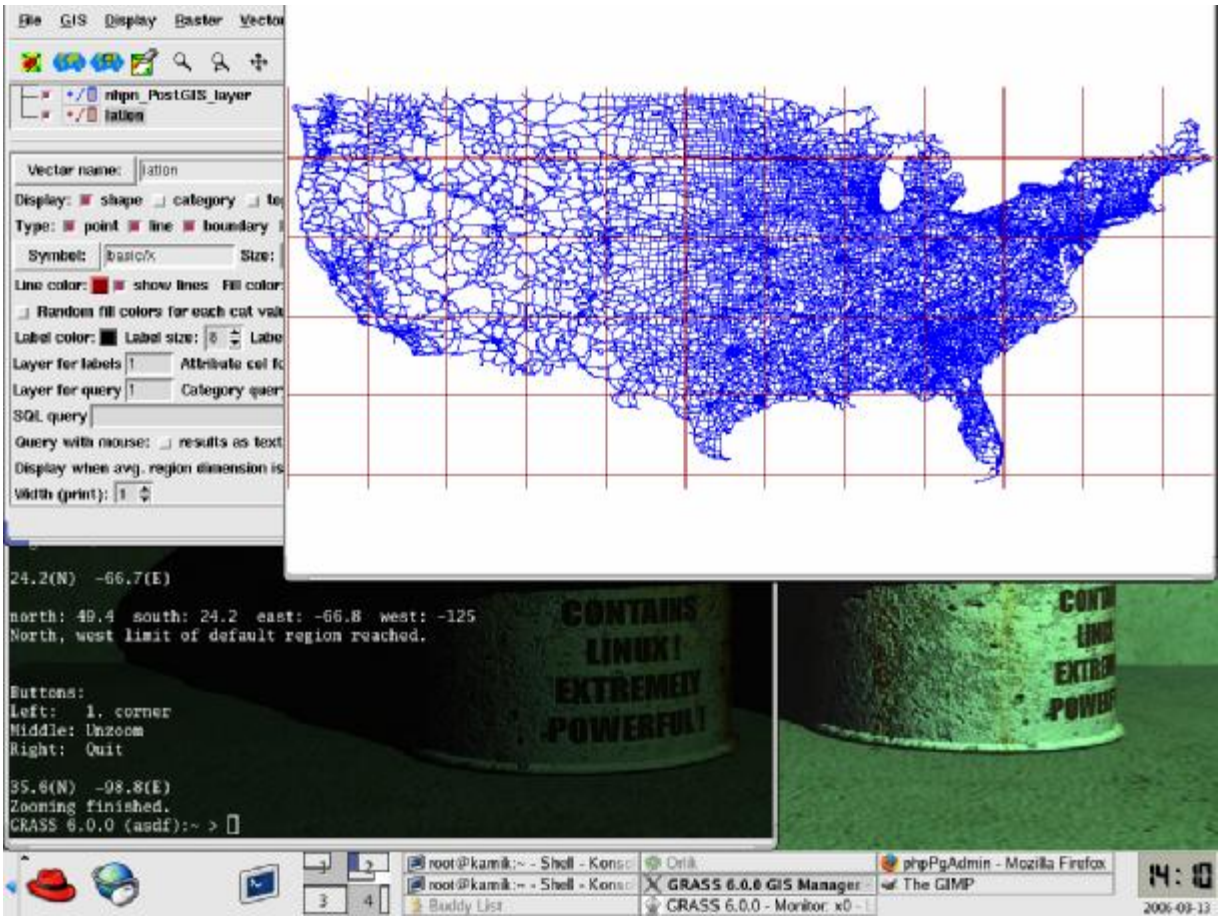
```
kam038@postgis:~$ shp2pgsql -W LATIN2 -s 102065 shp/provi.shp
provi_olomouc kam038 | psql -d kam038
```

9 PROGRAMOVÉ PROSTŘEDKY UMOŽŇUJÍCÍ PŘÍSTUP K DATŮM ULOŽENÝM V PostgreSQL/PostGIS

Programů, které mohou pracovat s vrstvami tvořenými tabulkami v PostgreSQL je celá řada (Thuban, JUMP, gvSIG), nicméně mnoho z nich je ve stádiu vývoje a zatím neobsahují složitější funkce pro modifikaci dat nebo prostorové analýzy. Existuje však také několik často používaných, silných nástrojů, které se mohou připojit k PostgreSQL/PostGIS. Tato kapitola se zabývá letmým popisem několika známých a také nově vznikajících, perspektivně se rozvíjejících programů, schopných připojení k databázi PostgreSQL/PostGIS.

9.1 GRASS

GRASS (Geographic Resources Analysis Support System) je open source GIS uvolněný pod licencí GNU GPL, umožňující práci s rastovými a vektorovými daty, tvorbu map, prostorové modelování a vizualizaci. Je určen pro OS UNIX, GNU/Linux, Mac OSX a je také možné jej nainstalovat na Windows pomocí Linux-like prostředí cygwin. Postup instalace je podrobně popsán v odkazu [15].



Obr. 13. Ukázka zobrazení dat v GRASS.

Od verze 6.0 je v programu GRASS implementována podpora pro import a export dat do PostGIS. Zdrojem dat pro GRASS není přímo tabulka uložená v databázi, ale před zobrazením je nutné importovat data do nativní podoby GRASSu. Stejně tak je nutné po požadovaných operacích a analýzách dat exportovat požadovanou vrstvu zpět do PostGIS. Níže uvádím příklady importu a dále exportu dat do GRASS z databáze PostgreSQL.

Import

```
GRASS 6.0.0 (cvicna):~ > v.in.ogr -o dsn='PG:host=postgis.vsb.cz
user=kam038 password=*** dbname=kam038' output=nhpn_me layer=nhpn
```

Export

```
GRASS 6.0.0 (cvicna):~ > v.out.ogr dsn='PG:host=postgis.vsb.cz
user=kam038 password=*** dbname=kam038' input=nhpn_me
olayer=nhpn_grass
```

Ukázka zobrazení importovaných dat silniční sítě USA v GRASS GIS z PostgreSQL/PostGIS je na obr. 13.

9.2 ArcGIS

Dalším velmi silným a rozšířeným nástrojem, který je bohužel určen výlučně pro platformu Windows je ArcGIS od firmy ESRI. Podporuje velké množství formátů dat včetně načítání, exportu a importu dat do PostgreSQL/PostGIS (ve verzi 9). Seznam formátů podporovaných ArcGIS je možné shlédnout na adrese [16].

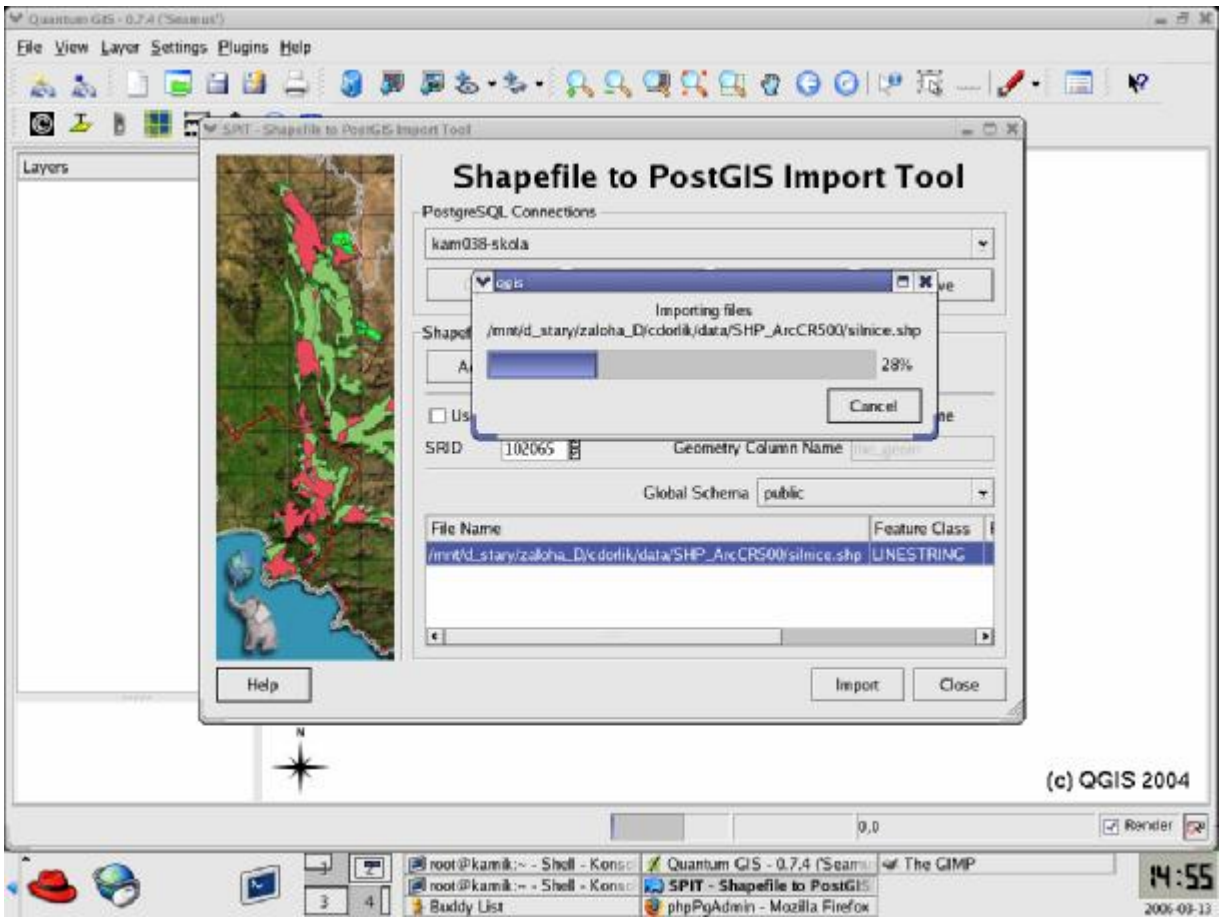
Přidání vrstvy z PostgreSQL/PostGIS do ArcMap se provádí pomocí ArcCatalog. V menu tools se vyberou extense a označí se položka „data interoperability“, čímž se rozšíří nabídka zdrojů o další zdroj dat.

ArcGIS umí načítat data z PostGIS verze 0.9 nebo nižší. Teprve od „verze“ extense Build 2182 (20051216) umí načítat i data z PostGIS verze 1.x. Tento problém je způsoben změnou interního formátu ukládání dat v PostgreSQL, neboť v nestabilní verzi PostGIS (verze 0.x) byla data ukládána ve formátu WKT, zatímco nyní jsou data interně ukládána ve formátu WKB.

9.3 QGIS

Quantum GIS je open source produkt fungující na OS GNU/Linux, UNIX, Mac OSX a Windows. Podporuje řadu vektorových a rastrových formátů dat a je možné s ním přímo

načítat vrstvy z PostgreSQL/PostGIS. Obsahuje funkce pro prohlížení, digitalizaci a import ESRI Shapefile do PostgreSQL/PostGIS (obr. 14).



Obr. 14. Ukázka importu Shapefile do PostgreSQL/PostGIS pomocí programu Quantum GIS.

9.4 uDIG

Jedná se o kvalitní open source prohlížečku a editor prostorových dat založený na eclipse. Jako zdroje dat mohou sloužit PostgreSQL/PostGIS, DB2, Web Feature Server, Web Map Server, ESRI Shapefile, a rastrové soubory. Na obrázku 15 uvádím ukázku vrstev načtených z WFS, PostgreSQL/PostGIS a ESRI Shapefile současně.

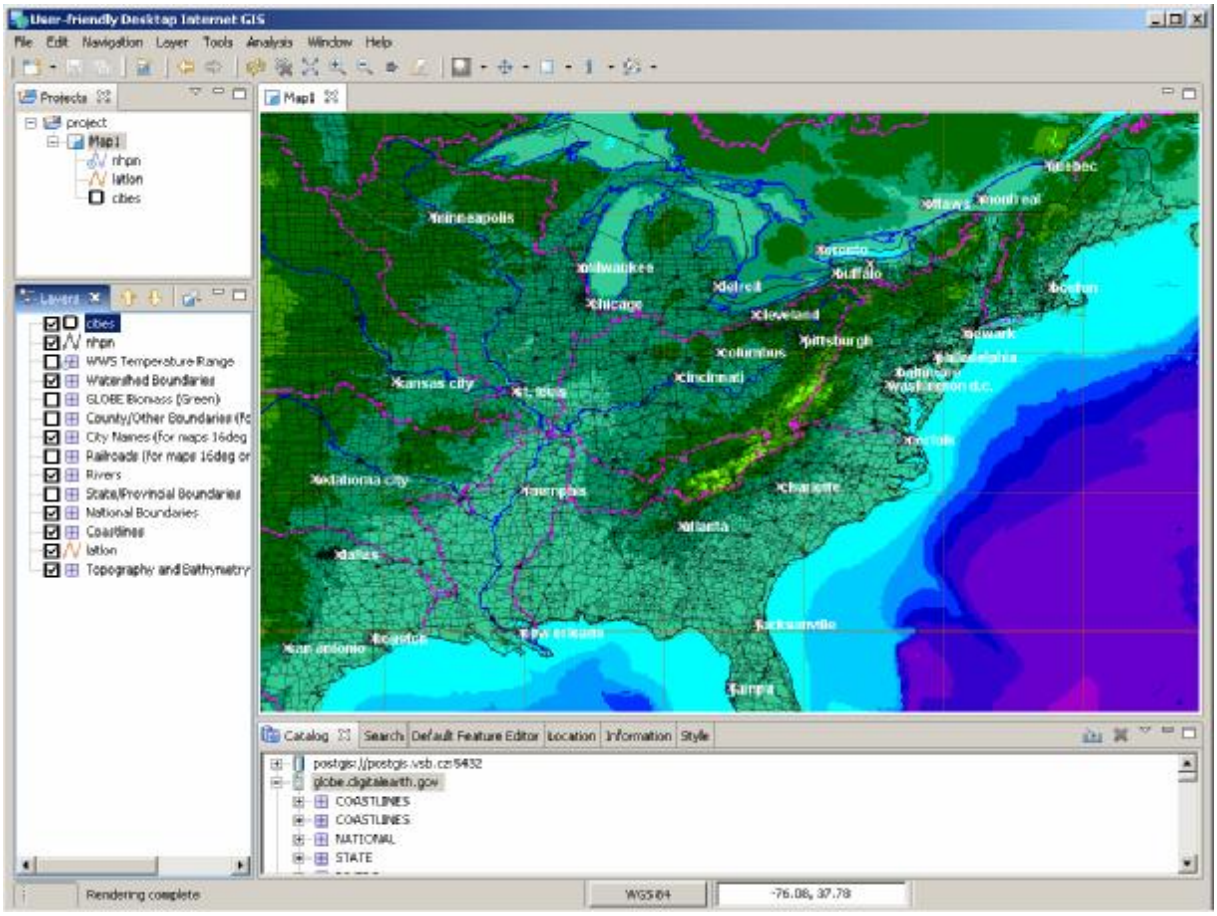
9.5 UMN Mapserver

Tento mapový server je open source produkt a je k dispozici buď jako CGI aplikace nebo DLL knihovna pro PHP. Aplikace umožňuje vytvářet rastrové obrázky ve formátech PNG nebo GIF z několika různých zdrojů prostorových dat (Oracle, MySQL, GeoTIFF, SHP, …) včetně PostgreSQL/PostGIS a podporuje specifikace WFS a WMS OpenGIS konsorcia.

Vzhled jednotlivých částí mapové kompozice (mapového pole, legenda, měřítko) se definuje v konfiguračním souboru s příponou *map*. Samotná mapová kompozice (tj. rozvržení komponent v mapě) se definuje v template souboru, což je v podstatě HTML soubor s parametry, které UMN MapServer nahrazuje hodnotami definovanými v *map* souboru. Níže uvádím ukázku přidání vrstvy *country* z PostGIS do mapové kompozice v *map* souboru:

```
...
LAYER
  NAME 'country'
  TYPE Polygon
  STATUS DEFAULT
  CONNECTIONTYPE postgis
  CONNECTION "dbname=kam038 user=kam038 password=***
host=postgis.vsb.cz"
  DATA "the_geom FROM country"
  COLOR 150 0 0
END
...
```

Je možno, aby řetězec uvedený za příkazem DATA obsahovat i složitý SQL dotaz, který může využívat prostorové funkce definované PostGISem.



Obr. 15. Ukázka načtení mapových vrstev ze zdrojů WFS, PostgreSQL/PostGIS a ESRI Shapefile v programu uDIG.

10 NĚKTERÉ FUNKCE PRO ANALÝZU DAT V PostGIS

PostGIS obsahuje velkou řadu funkcí, pomocí nichž je možno provádět různé operace s objekty. Funkce mohou být typu boolean nebo constructive. Seznam běžně používaných funkcí je možné nalézt na stránkách [18]

10.1 Funkce typu boolean

Funkce typu boolean vrací hodnotu typu true/false (1/0). Dále uvádím několik základních funkcí i s příklady použití.

Funkce Touches(geometry, geometry)

Touches() testuje, zda se dva objekty dotýkají. Její použití uvádím v následující ukázkové funkci, která prochází všechny linie v tabulce *line* a vypisuje ostatní linie, které na ni navazují:

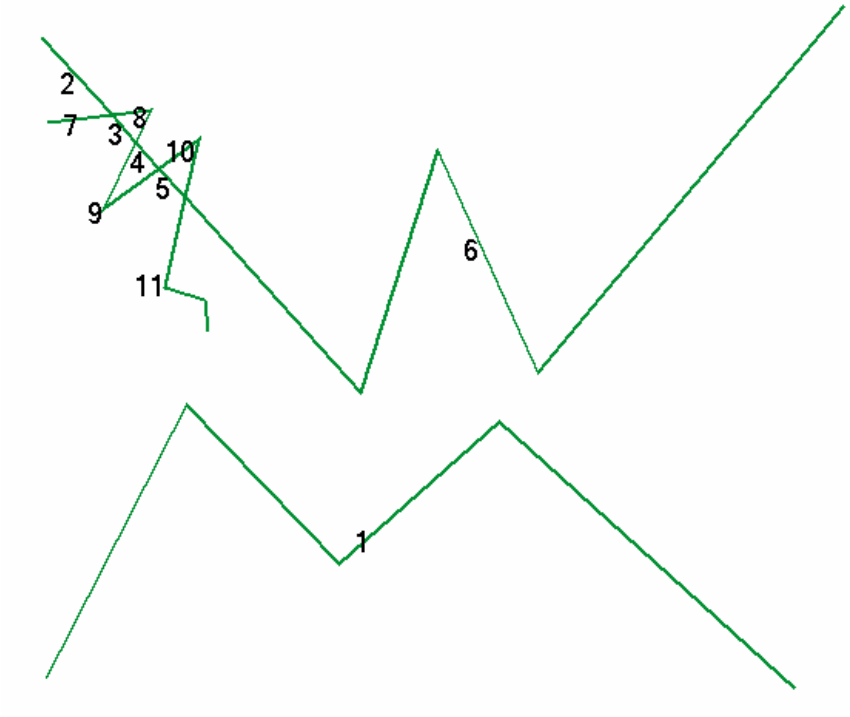
```
CREATE OR REPLACE FUNCTION incid() RETURNS void AS $$
DECLARE
  a RECORD;
  b RECORD;
  c boolean;

BEGIN
  FOR a IN SELECT id FROM public.line LOOP
    FOR b IN SELECT id FROM public.line WHERE id <> a.id LOOP
      SELECT INTO c touches((SELECT the_geom FROM line WHERE
id=a.id),(SELECT the_geom FROM line WHERE id=b.id));
      IF c = true THEN
        RAISE NOTICE 'linie % navazuje na %', a.id, b.id;
      END IF;
    END LOOP;
  END LOOP;
END;
$$ LANGUAGE PLPGSQL;
```

Výstup z funkce je následující:

```
kam038=# SELECT incid();
NOTICE:  linie 2 navazuje na 3
NOTICE:  linie 2 navazuje na 7
NOTICE:  linie 2 navazuje na 8
NOTICE:  linie 3 navazuje na 2
NOTICE:  linie 3 navazuje na 4
NOTICE:  linie 3 navazuje na 7
NOTICE:  linie 3 navazuje na 8
NOTICE:  linie 3 navazuje na 9
NOTICE:  linie 4 navazuje na 3
NOTICE:  linie 4 navazuje na 5
NOTICE:  linie 4 navazuje na 8
NOTICE:  linie 4 navazuje na 9
```

```
NOTICE:  linie 4 navazuje na 10
NOTICE:  linie 5 navazuje na 4
NOTICE:  linie 5 navazuje na 6
NOTICE:  linie 5 navazuje na 9
NOTICE:  linie 5 navazuje na 10
NOTICE:  linie 5 navazuje na 11
NOTICE:  linie 6 navazuje na 5
NOTICE:  linie 6 navazuje na 10
NOTICE:  linie 6 ...
...
...
```

Uvedená funkce byla použita na liniích zobrazených na obrázku 16.



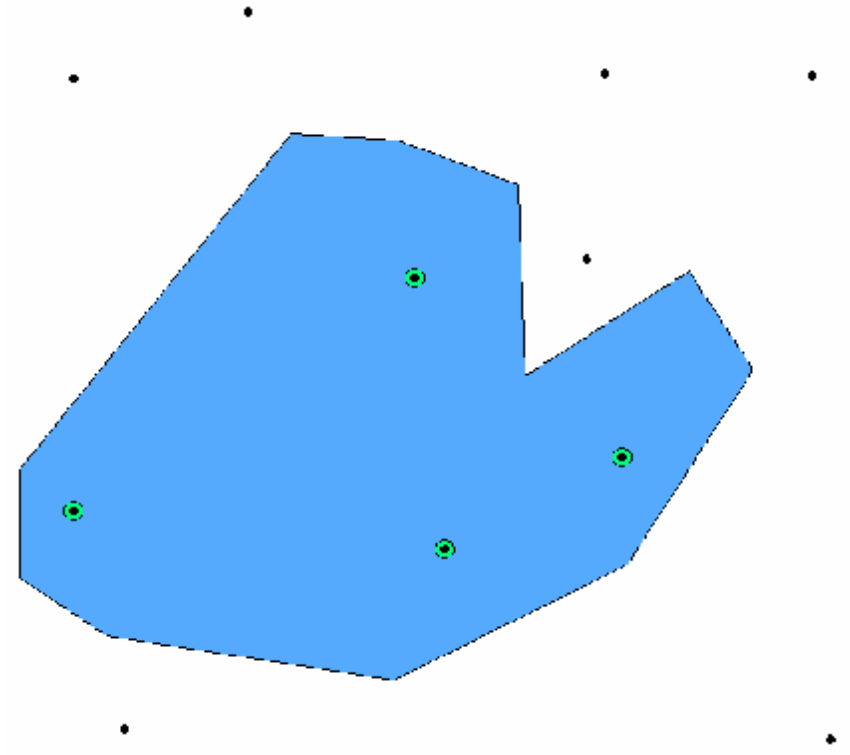
Obr. 16. Linie použité při ukázce funkce *Touches()*.

Funkce Within(geometryA, geometryB)

Within() testuje zda geoprvek A je obsažen v geoprvku B. *Within()* je použito k vytvoření nové vrstvy z bodů, který jsou obsaženy v polygonu.

```
CREATE TABLE within AS SELECT * FROM body WHERE
Within(the_geom,(SELECT the_geom FROM polygon))
```

Výsledek je zobrazen zelenými body na obr. 17



Obr. 17. Ukázka použití funkce *Within()*.

10.2 Funkce typu constructive.

Constructive funkce vrací nově vygenerovaný objekt na základě vložených objektů (sjednocení, rozdíl, …) nebo případně také mohou vracet číselné hodnoty (délka linií, plocha polygonu, počet bodů tvořících linii, …).

Funkce Length(geometry)

Length() vrací délku geoprvku v projekčních jednotkách. V následujícím příkladě je použita pro výpočet celkové délky silnic v ČR.

```
kam038=# SELECT sum(length(the_geom))/1000 AS delka_silnic_km FROM
silniceCr;
delka_silnic_km
------------------
38948.4775410676
(1 row)
```

Funkce Area(geometry)

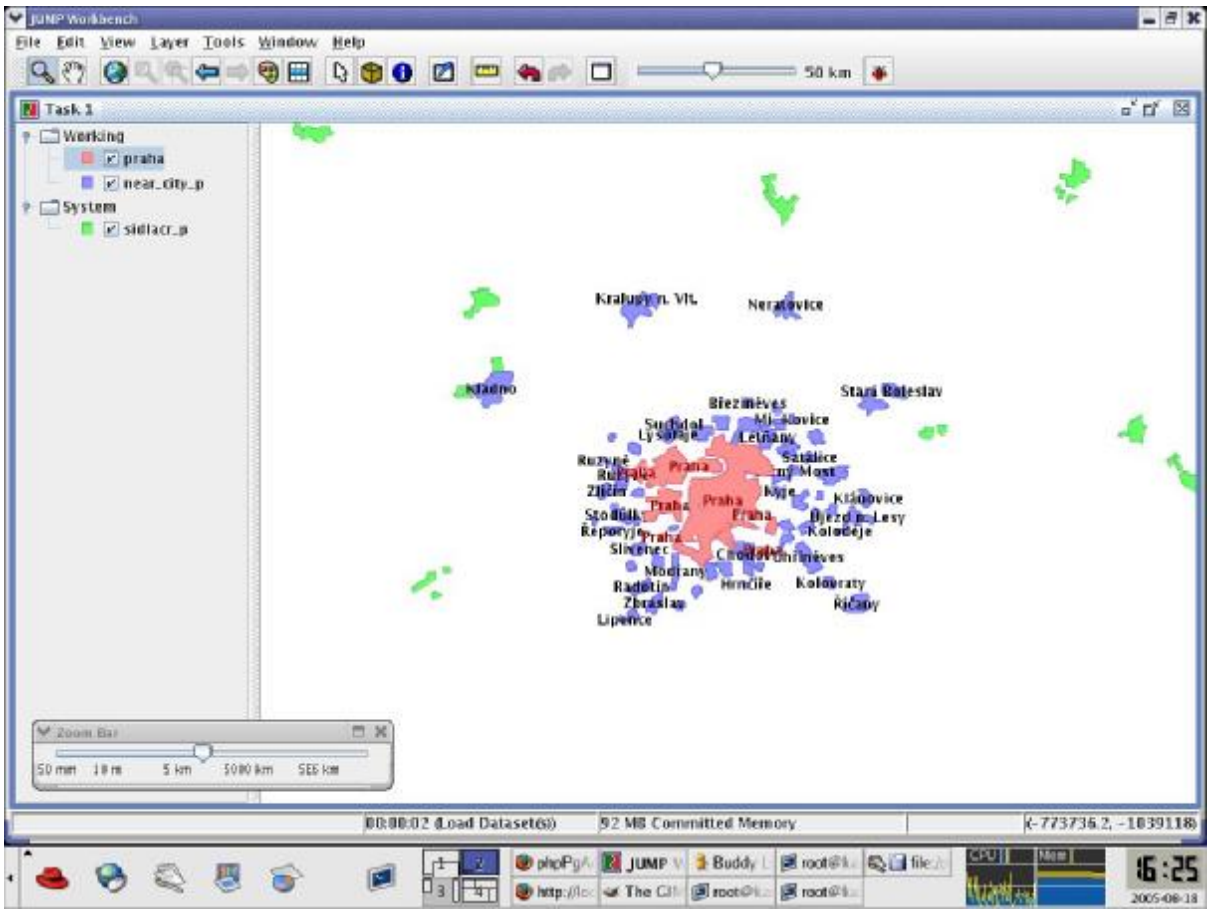
Funkce *Area()* vrací plochu geoprvku. V následujícím příkladu jejího použití je pomocí ní vypočteno procentuální zastoupení lesních ploch z celkové plochy okresu Jeseník.

```
kam038=# SELECT Sum(Area(Intersection(o.the_geom,
l.the_geom)))/(SELECT Area(the_geom) FROM okresy WHERE
name_utf8='jeseník')*100 AS procenta_zalesneni FROM okresy AS o,
lesy AS l WHERE o.the_geom && l.the_geom AND o.name_utf8 =
'jeseník';
procenta_zalesneni
------------------
54.824044186251
```

Funkce Distance(geometry, geometry)

Funkce *Distance()* vrací nejkratší vzdálenost mezi objekty. Příklad použití funkce *Distance()* vybírá města ve vzdálenosti 15 km od Prahy (jsou zvýrazněny modře). Výsledek je zobrazen na obrázku 18.

```
CREATE TABLE near_city_p (nazev varchar(30), the_geom geometry);
CREATE TABLE praha AS SELECT nazev, the_geom FROM sidlacr_p WHERE
nazev='Praha';
INSERT INTO near_city_p (nazev, the_geom) SELECT nazev, the_geom
FROM sidlacr_p WHERE distance( the_geom,  praha.the_geom) < 15000;
```



Obr. 18. Ukázka použití funkce *Distance()*.

Funkce Extent (geometry set)

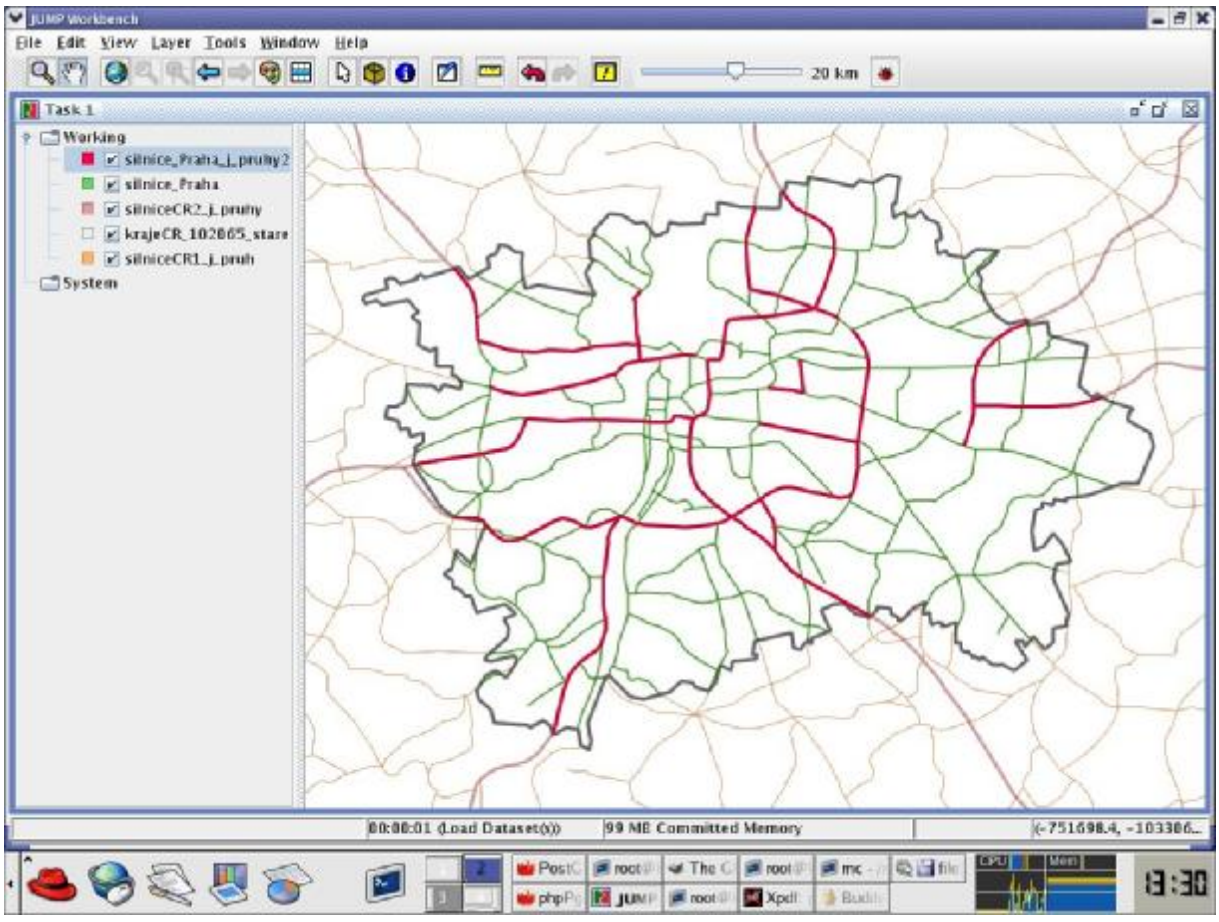
Extent() vrací datový typ BOX3D, který reprezentuje maximální a minimální rozsah souřadnic daného seznamu geoprvků.

```
kam038=# SELECT extent(the_geom) FROM krajecr_102065;
extent
----------
BOX(-904539.625 -1227663,-431680.59375 -935232.3125)
```

Funkce Intersection (geometry, geometry)

Funkce *Intersection()* vrací datový typ geometry představující místa průniku dvou geoprvků. Níže uvedený příklad použití vytvoří tabulku se silnicemi s dvěmi jízdními pruhy v Praze. Obrázek 19 byl vytvořen pomocí několika podobných příkazů.

```
kam038=# CREATE TABLE silnice_Praha_j_pruhy2 as SELECT
intersection(r.the_geom, m.the_geom) AS intersection_geom, r.* FROM
krajeCR_102065_stare AS r, silnicecr102065 AS m WHERE r.the_geom &&
m.the_geom AND r.nazev = 'Hl.m. Praha' AND m.j_pruhy=2;
```

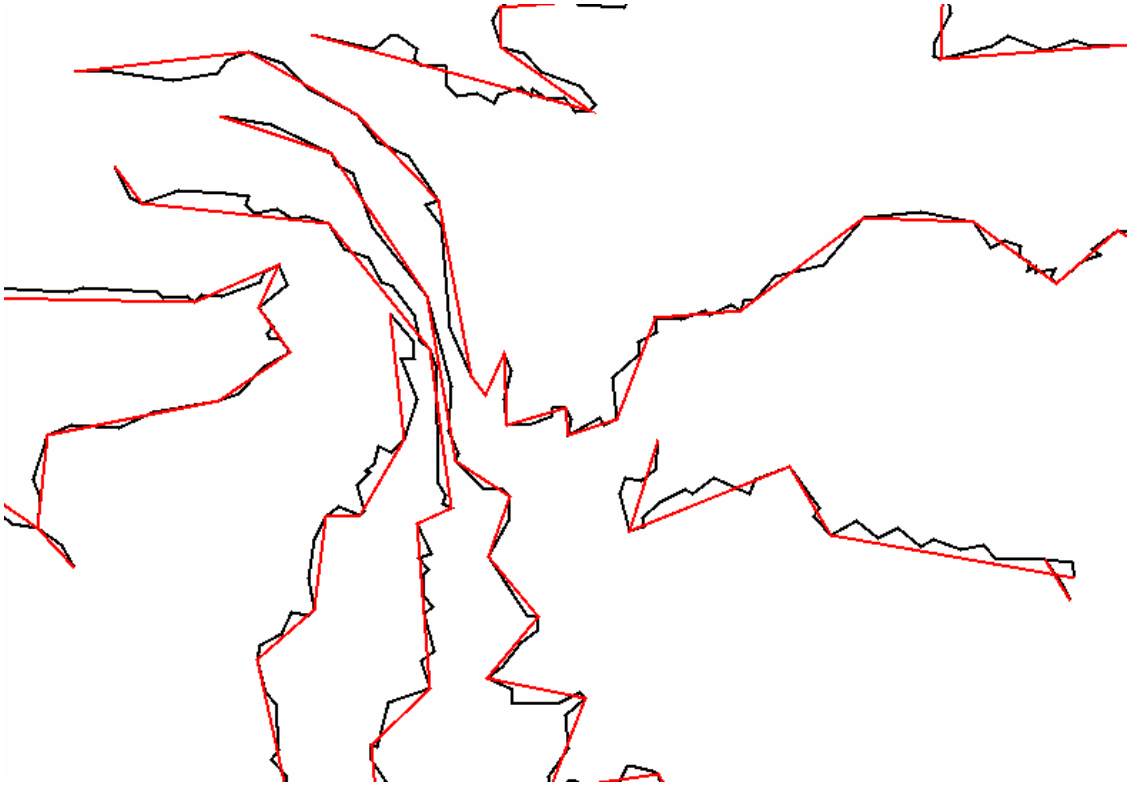


Obr. 19. Ukázka použití funkce *Intersection()*.

Funkce Simlify(geometry, tolerance)

Tato funkce provádí generalizaci daných geoprvků. Na obr. 20 jsou černě znázorněny původní linie a červeně jsou znázorněny generalizované linie. Parametr *tolerance* představuje maximální povolenou kolmou vzdálenost mezi generalizovaným bodem a generalizující linií. Udává se v projekčních jednotkách.

```
kam038=# CREATE TABLE simplified_rivers AS SELECT
simplify(the_geom,0.5) FROM rivers;
```



Obr. 20. Ukázka funkce *Simplify()*.

11 ZÁVĚR

V úvodu jsou popsány některé výhody, které má ukládání prostorových a popisných atributů do jedné tabulky pomocí relačních databází. Hlavními výhodami je centralizace dat, snadné řízení přístupu uživatelů pomocí definování přístupových práv k datům, sjednocení popisných a prostorových atributů včetně definice souřadnicového systému, stálá aktuálnost dat a v neposlední řadě možnost využití funkcí definovaných PostgreSQL/PostGIS k složitým operacím s daty, včetně transformace mezi souřadnicovými systémy.

Dále se práce krátce věnuje porovnání možností PostgreSQL/PostGIS a ostatních významných SŘBD (Oracle Spatial, Informix a MySQL) schopných ukládat prostorová data. Následuje seznámení s extensí PostGIS a SŘBD PostgreSQL, popis jejích možností a vlastností. Je popsán postup instalace SŘBD PostgreSQL/PostGIS na OS GNU/Linux a Windows. Práce se také věnuje tvorbě uživatelů, databází a nejběžnějším textovým i grafickým nástrojům používaným k administraci databáze.

V další části jsou popsány některé nejvýznamnější funkce RDBMS PostgreSQL použitelné při tvorbě a administraci datového skladu. Především je věnována pozornost indexaci databáze, tvorbě triggerů a uživatelem definovaných funkcí v PL/pgSQL.

Závěrem byla provedena ukázka použití několika funkcí definovaných extensí PostGIS na prostorových datech.

Byla ověřována bezchybnost práce databáze v oblasti opakovaného importu, exportu a převodů objektů mezi různými kartografickými zobrazeními. Také byla ověřována účinnost ochrany dat nastavováním přístupových práv k tabulkám různým uživatelům a skupinám. Byly prováděny převody uživatelů mezi skupinami a tvorba nových uživatelů. Následně byly činěny pokusy o přístup k tabulkám. Při všech operacích bylo dosaženo očekávaných výsledků.

SEZNAM POUŽITÉ LITERATURY

[1] GIS GRASS a JTSK (Krovak), *http://mpa.itc.it/radim/jtsk/index.html*

[2] Často kladené dotazy (FAQ) PostgreSQL, *http://postgresql.ok.cz/FAQ2.html#1.2*

[3] MySQL vs PostgreSQL vs Firebird II, *http://www.root.cz/clanky/mysql-vs-postgresql-vs-firebird-ii/*

[4] PostgreSQL, FTP browser, *http://www.postgresql.org/ftp/binary/v8.1.3/win32/*

[5] DC Maintenance Management System, *http://dcmms.sourceforge.net/index.php*

[6] PostgreSQL Installer, *http://pginstaller.projects.postgresql.org/*

[7] What is phpPgAdmin?, *http://phppgadmin.sourceforge.net*

[8] PgAdmin, postgresql tool, *http://www.pgadmin.org/*

[9] Internetový portál zabývající se problematikou Linuxu, *http://www.linuxsoft.cz/*

[10] PostgreSQL manual, vacuum, *http://www.postgresql.org/docs/8.1/interactive/sql-vacuum.html*

[11] PostgreSQL maual, seznam vyjimek, *http://www.postgresql.org/docs/8.1/interactive/errcodes-appendix.html*

[12] PostgreSQL 12 - urychlení výběrů, *http://www.linuxsoft.cz/article.php?id_article=878*

[13] PostgreSQL manual, multicolumn indexes, *http://www.postgresql.org/docs/8.1/interactive/indexes-multicolumn.html*

[14] FWTools: Open Source GIS Binary Kit for Windows and Linux, *http://fwtools.maptools.org/*

[15] GRASS, binary distribution for cygwin, *http://geni.ath.cx/grass.html*

[16] ESRI, ArcGIS Data Interoperability Extension – Supported Formats, *http://www.esri.com/software/arcgis/extensions/datainteroperability/about/supported-formats.pdf*

[17] PostgreSQL manual, string functions and operators, *http://www.postgresql.org/docs/current/static/functions-string.html*

[18] PostGIS, OpenGIS functions, *http://postgis.refractions.net/docs/ch06.html*

SEZNAM OBRÁZKŮ