Školení PostGIS pro začátečníky

verze 0.6beta

GISMentors

25.09.2019

Obsah

PostGIS je rozšíření objektově-relačního open source databázového systému PostgreSQL umožňující uložení, správu a analýzu geografických dat. PostGIS implementuje v prostředí PostgreSQL specifikaci Simple Features konsorcia Open Geospatial Consortium.

Obr. 1: Logo projektu PostGIS

PostGIS lze použít jako databázové uložiště dat společně s oblíbeným desktopovým open source GISem QGIS.

PostGIS je podobně jako QGIS multiplatformní a plně funkční na platformách jako GNU/Linux, MS Windows či Mac OSX.

Poznámka k datové sadě GISMentors

Datová sada je stažitelná pro PostgreSQL ve formátu dump (595 MB). Další informace v kapitole *Poznámky k instalaci a obnově databáze*. Datová sada GISMentors je založena na datech pocházejících pouze z otevřených či veřejných zdrojů jako je EU-DEM, RÚIAN, OpenStreetMap, Dibavod, AOPK a IPR.

Varování: Toto je pracovní verze školení, která je aktuálně ve vývoji!

Vstupní znalost

- Uživatel má základní znalosti GIS

Výstupní dovednost

- Uživatel získá základní znalost jazyka SQL
- Uživatel je schopen data z databáze v prostředí QGIS vizualizovat a editovat
- Uživatel je schopen z prostředí QGIS provádět v databázi jednodušší prostorové dotazy a další základní operace
- Uživatel zvládne naimportovat do databáze vlastní data a dále je exportovat z databáze do ostatních GIS formátů

Požadavky

- PostgreSQL, volitelně pgAdmin, LibreOffice
- QGIS 2.14 a vyšší
- PostGIS 2.0 a vyšší

KAPITOLA 1

Úvod

1.1 Databáze

1.1.1 Kdy ukládat data do databáze?

V geoinformatické praxi pracujeme se třemi typy zdrojů dat. V prvé řadě se jedná o data uložená v souborovém systému. Hovoříme-li o vektorových datech, tak může jít typicky o data v zastaralém, leč stále nejpoužívanějším formátu Esri Shapefile, případně OGC GML. Dalším typem jsou webové služby, jmenovitě OGC WFS (Web Feature Service). V případě WFS si aplikace vyžádá pomocí souboru ve značkovacím jazyce XML data na vzdáleném serveru po síti. Posledním typem uložení dat, kterému je věnováno toto školení, je databáze. Většina současných databází, ať již open source nebo ryze proprietárních, podporuje v nějaké míře ukládání a dotazování prostorových prvků. Ať už MySQL, Oracle, nebo MSSQL a v neposlední řadě PostgreSQL, kterému je věnován tento kurz.

Poznámka pro pokročilé

Hranice mezi jednotlivými typy zdrojů dat nemusí být vždy jednoznačná. Existují například takzvané *souborové databáze*, tedy soubory, které se chovají podobným způsobem jako databázový server, ovšem bez řady výhod, které poskytuje plnohodnotný databázový systém. Na druhou stranu se s nimi o poznání snáze manipuluje. Příkladem může být MS Access nebo open source SQLite (a jeho prostorové nadstavby OGC GeoPackage a SpatiaLite).

Provoz databáze přináší ve srovnání s daty v souborech určité požadavky na režii. O její správu a nastavení se musí starat kvalifikovaný specialista, má určité nároky na hardware apod. Co nám tedy přináší a kdy je pro nás nezastupitelná?

V první řadě je třeba vzít v potaz objem dat. Od jistého objemu není možné efektivně pracovat s daty uloženými v souborech. Naproti tomu v databázi můžeme pomocí indexů přistupovat přímo k jednotlivým

záznamům tak, jak jsou uloženy na datových stránkách.

Referenční integrita

Další benefit, který nám databáze může přinést je „hlídání“ tzv. *referenční integrity*.

Referenční integrita znamená, že tabulky jsou mezi sebou provázány cizími klíči. Tedy pokud podřízená (slave) tabulka obsahuje položku s odkazy do jiné *nadřízené* tabulky, není možné do podřízené tabulky přidat záznam, pokud v nadřízené tabulce neexistuje hodnota, na kterou odkazuje cizí klíč. Nemůžeme tedy například do tabulky jednotlivých vozidel přidat vozidlo s odkazem na typ *tříkolka*, pokud nemáme v tabulce typů vozidel typ *tříkolka*. Nebo pokud máme tabulku staveb a parcel, při správně nastavené referenční integritě nám databáze nedovolí vložit budovu na neexistující parcele a pod.

Další užitečnou vlastnosti je možnost nastavit chování podřízeného záznamu při smazání souvisejícího záznamu v nadřízené tabulce. Můžeme zvolit RESTRICT nebo CASCADE. V případě CASCADE se související záznamy mažou, v případě RESTRICT není možné nadřízený záznam smazat, dokud jsou na něj navázány záznamy v podřízených tabulkách.

Spolupráce

Není obvyklé, aby k jednomu souboru přistupovalo více klientských aplikací zároveň, protože by si ho přepisovaly „pod rukama“. Databáze je v tomhle daleko pokročilejší a umožňuje, aby nad jednou datovou sadou mohlo pracovat více klientů najednou. V databázi je navíc možné nastavovat práva na zápis, čtení a manipulaci s tabulkami, schématy, funkcemi… Podobně jako v souborovém systému.

Transakce

Transakčnost databáze znamená, že se série změn provede buď celá nebo vůbec. Typická (a tím pádem pěkně otřepaný případ) je situace, kdy převádíme peníze z účtu na účet. Tedy, nebylo by dobré, aby byly z jednoho účtu peníze odečteny, aniž by na cílový účet byly přidány.

Seznam požadavků na transakční databázi bývá označován zkratkou ACID. Znamená to *Atomic, Consistent, Isolated, Durable*. Znamená to, že transakce je nedělitelná, před i po jejím proběhnutí musí být platná referenční integrita, transakce se navzájem neovlivňují a změny jsou trvalé i po případné havárii databázového serveru.

1.2 Co je databáze?

Databázi, ať už relační nebo dokumentovou, si můžeme představit jako knihovnu. V knihách (tabulkách) máme nějaké informace. Informace pro nás vyhledávají knihovnice (obslužné programy). K tomu používají katalogy a rejstříky (indexy). Organizace knihovny je plně pod naší kontrolou, ovlivňujeme hardware (kolik bude mít budova pater (disků), kolik bude volných regálů a manipulačního prostoru atd.), kolik bude mít knihovna fyzických zaměstnanců (počet jader procesoru). Dále ovlivňujeme organizaci, budou knihy řazeny podle abecedy podle názvů, podle klíčových slov, podle jména autora? Jak často budeme aktualizovat katalogy a rejstříky (aktualizovat indexy)? Kolik místa vlastně na katalogy/indexy vyhradíme? Jak

budeme nakládat s místem po vyřazených svazcích (proces VACUUM)? A tak dále. Se svými zaměstnanci komunikujeme v jazyce *SQL* (pokud tedy hovoříme o relační databázi).

1.2.1 Tabulky

V relační databázi ukládáme data do tabulek (tzv. relací). Tabulka je svisle dělena na jednotlivé sloupce (často označovány jako atributy nebo položky) a vodorovně na řádky (záznamy). Data v jednom sloupci musí mít stejný *datový typ* (datum, celé číslo, číslo s plovoucí desetinnou čárkou, textový řetězec apod.).

1.2.2 Schémata

Schémata můžeme vnímat podobně jako adresářovou strukturu, ovšem bez možnosti dalšího zanořování, případně jako *jmenný prostor*. Umožňuje nám logicky dělit databázi, což oceníme například při zálohování, při nastavování práv. Databázové tabulky, funkce, indexy apod. musí mít v rámci schématu (schéma je možné vnímat jako součást názvu) unikátní název. Tudíž můžeme mít v databázi stejně pojmenované tabulky v různých schématech. Příklad využití je například při databázi rozdělené do schémat geograficky. Další výhodné využití je při historizování záznamů, kdy máme schéma *historie* s podobnou strukturou jako schéma s platnými daty.

1.2.3 Datové typy

Datové typy odpovídají typům z programovacích jazyků typu C. Základem jsou celočíselné typy (*integer*, *bigint* apod.) a řetězce (*varchar*, *char*, *text* ...), tím ovšem výčet zdaleka nekončí. Pro prostorovou reprezentaci používáme datový typ *geometry* nebo *geography*. Záznamu v tabulce odpovídají kompozitní typy, celé datové struktury je možné ukládat do *nerelačních datových typů* jako je JSON, hstore nebo XML a dalo by se dále pokračovat.

1.2.4 Indexy

Indexy v databázi slouží k co možná nejrychlejšímu dohledání záznamů v tabulce. Fungují na podobném principu jako rejstřík v knize. Jedná se o jakýsi utříděný seznam klíčů spojených s odkazem na konkrétní datovou stránku, na místo na pevném disku, kde je uložena požadovaná informace. Smyslem indexu je provést při dohledání záznamu minimum porovnání hodnot v indexu s požadovanou hodnotou. U neindexované tabulky bychom museli porovnat požadovanou hodnotu se všemi záznamy.

Poznámka pro pokročilé

Nejčastějším typem indexu je B-tree, zde jsou hodnoty uloženy ve stromovité struktuře založené na dichotomickém větvení. Na každém uzlu porovnáme požadovanou hodnotu s hodnotou na uzlu a zjistíme, jestli je větší nebo menší. S každým patrem je síto jemnější. To je velice efektivní, když si uvědomíme, že při zdvojnásobení objemu dat přibude jen jedno porovnání navíc. B-tree index je možné sestavit jen nad položkami s takovým typem dat, který je možné porovnávat pomocí operátorů < a >. Nehodí se tedy pro data vícedimenzionální, např. prostorová data.

1.2.5 Omezení (constraints)

V odstavci věnovaném referenční integritě je zmíněno, že není možné vložit do sloupce s cizím klíčem hodnotu, která není v *nadřízené* tabulce. To je příkladem *omezení cizího klíče*. Dalším častým příkladem je omezení na unikátní hodnotu, což je podmínka pro *primární klíč*, tedy hodnotu, podle které je možné jednoznačně identifikovat záznam v tabulce. Omezení ovšem můžeme vytvářet dle libosti, například můžeme v tabulce osob nastavit, že není možné do sloupce se jménem vložit jméno *František*, případně do nějakého číselného sloupce hodnotu, která není dělitelná jedenácti, geometrii s rozlohou větší než hektar apod.

Zde je dobré si uvědomit, že pokud se pokusíte vložit data do sloupce a porušíte omezení, vrátí server chybu. Pokud tedy bude tato dávka součástí transakce, neprovede se celá transakce.

1.2.6 Pohledy (views)

Pohledy jsou uložené dotazy, které se chovají obdobně jako tabulky. Můžeme je dotazovat, nastavovat jim práva. K tabulkám, do kterých pohledy nahlížejí, přistupují s právy toho, kdo je vytvořil. Můžeme tedy pohledem zpřístupnit pro některé uživatele vybraný obsah tabulek, které sami nevidí.

Specifickou záležitostí jsou materializované pohledy. Zde je výstup dotazu uložen do tabulky a zároveň je uložen dotaz, kterým byl materializovaný pohledy vygenerován. Proto může být snadno přegenerován příkazem REFRESH MATERIALIZE VIEW.

Poznámka: Materializované pohledy podporuje PostgreSQL od verze 9.3.

1.2.7 Triggery

Trigger, neboli spoušť spustí proceduru při nějaké události. Existují dva základní typy triggerů a to *DML* a *DDL* triggery.

DML, tedy *Data Manipulation Language* trigger se spustí při manipulaci s daty, tedy při vložení, smazání, případně aktualizaci záznamu. Obvyklé využití je například archivování smazávaných hodnot, kontrolu dat při vstupu a podobně. Pomocí triggerů lze ošetřit také kontrolu podobně jako u omezení. Pokud nastavíme trigger tak, aby se spustil před vložením záznamu, můžeme eliminovat duplicitní záznamy, dříve než dojde k chybě a tím pádem nedojde k pádu transakce.

DDL, tedy *Data Definition Language* trigger je v PostgreSQL relativně čerstvá novinka a spustí se při změně ve struktuře, například při přidání tabulky může nastavit práva, replikace apod.

Obdobou triggerů jsou pravidla, ta ovšem nedisponují všemi možnostmi triggerů a nedoporučuje se jich příliš používat. Nicméně občas se mohou hodit, pokud chceme pracovat s pohledem jako s tabulkou a nastavit, co se má dít při vkládání nebo manipulaci s daty.

1.2.8 Funkce

Funkce je v databázi uložená procedura, kterou spustíme dotazem. V PostgreSQL může být napsaná v jazyce SQL, v procedurálním jazyce PosgreSQL PL/pgSQL plpgsql či v dalším z jazyků, které PostgreSQL

podporuje jako je Python, Perl, R, Javascript a další. Případně může být importovaná z externího modulu napsaného například v jazyce C.

Funkce tedy spouští nějaký kód, může vracet jednu hodnotu, jednu hodnotu z více záznamů (agregační funkce), případně může vracet více záznamů, nebo provádět nějaké změny v databázi (například funkce PostGISu AddGeometryColumn). Specifickou skupinou jsou analytické window funkce.

Nastavování práv k funkcím je složitější než u pohledů, je možno nastavit SECURITY DEFINER práva a potom přistupuje funkce k tabulkám s právy svého tvůrce.

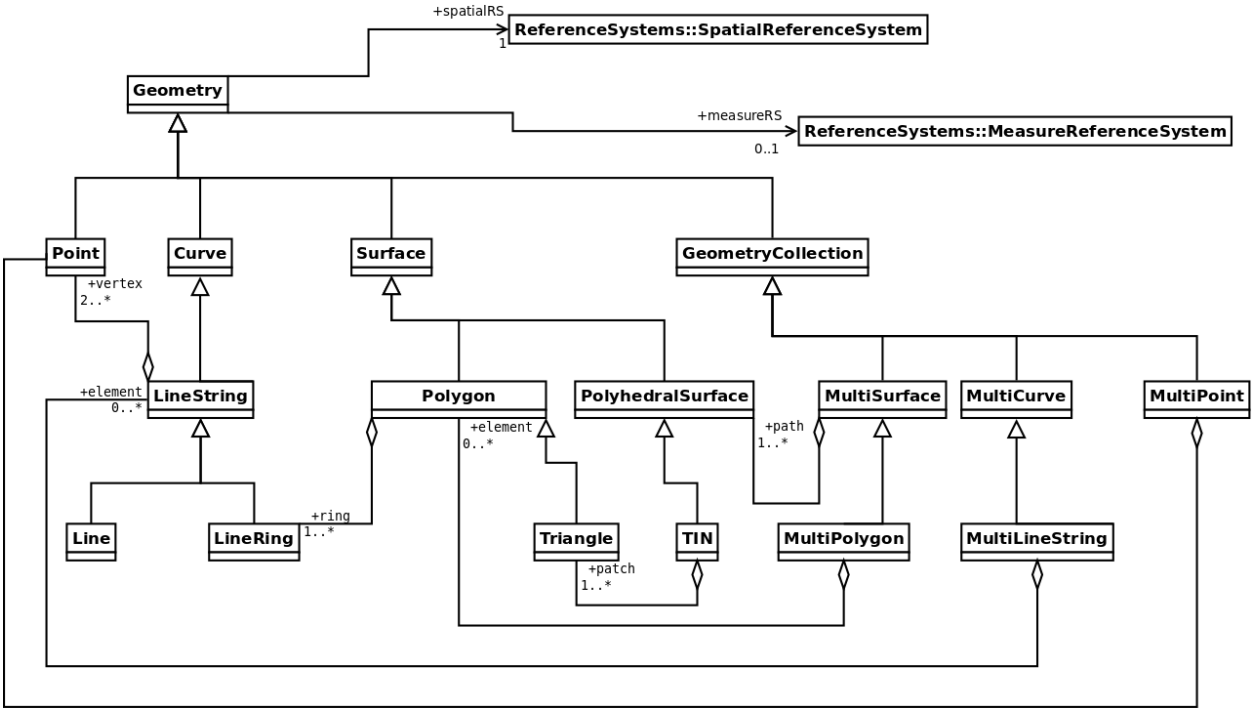
1.3 A co prostorová databáze?

Prostorová databáze se podobá takové knihovně, ve které jsou kromě knih také mapy, atlasy, globusy... Zkrátka nosiče informací, které zaznamenávají také umístění jednotlivých údajů.

PostGIS používá pro popis prostorových objektů specifikaci OGC Simple Features Access (SFA). Tato specifikace popisuje společnou architekturu pro *tzv. jednoduché geoprvky* a specifikuje jejich uložení v digitální podobě.

Poznámka: V roce 2004 byla specifikace OGC SFA přijata jako mezinárodní norma označovaná jako ISO 19125 a později v roce 2006 adoptována jako technická norma ČSN 19125.

Specifikace OGC SFA zavádí pro popis geometrie geoprvků nové datové typy jako je např. *Point*, *LineString*, *Polygon* a další.



Obr. 1.1: Přehled jednotlivých typů geometrie dle specifikace OGC SFA.

KAPITOLA 2

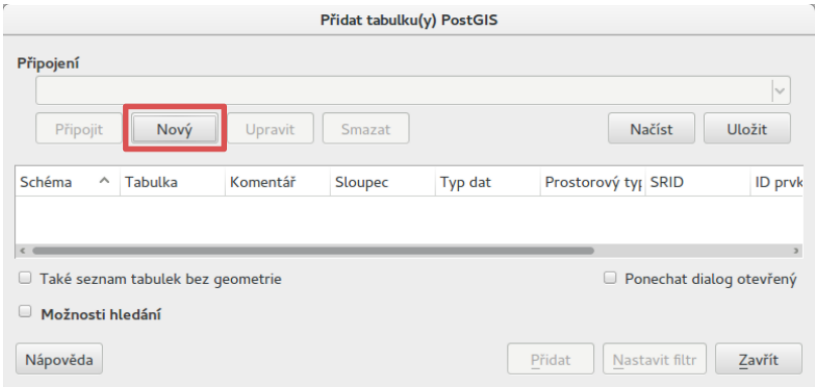
Začínáme

Na úvod si ukážeme přístup k datům uložených v databázi z prostředí desktopového programu QGIS.

Poznámka: Více o tomto programu se dozvíte na školení QGIS pro začátečníky.

2.1 Zobrazujeme data v QGIS

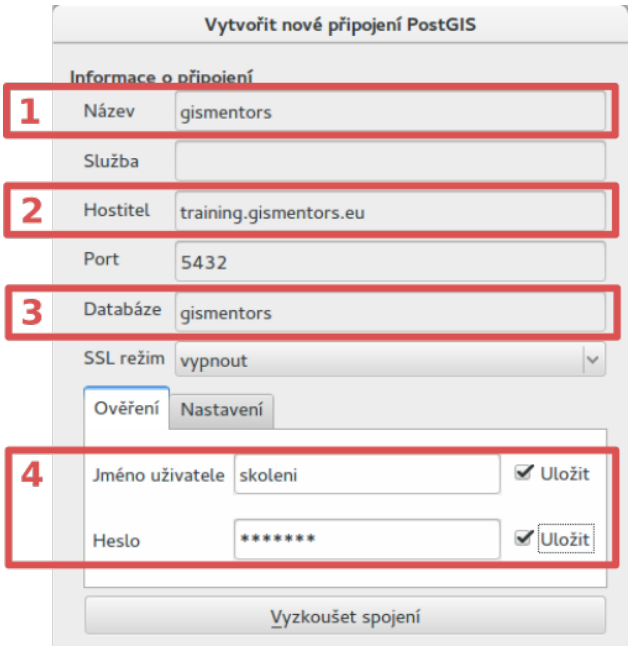
Vektorová data uložená v geodatabázi PostGIS je možné načíst buď z menu *Vrstva* → *Přidat vrstvy* → *Přidat vrstvu PostGIS* anebo z *nástrojové lišty* aplikace QGIS  . Další možností je použít *datový prohlížeč*. Objeví se dialog, ve kterém definujeme parametry *nového* připojení k databázi.



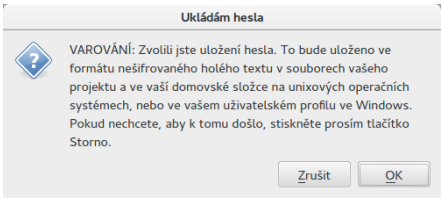
Nastavíme:

- název spojení (1)
- hostitel (adresa serveru, pokud je to localhost, nemusíme vyplňovat) (2)

- databáze, ke které se chceme připojit (3)
- uživatelské jméno a heslo pro připojení k databázi (4)



Poznámka: Při opětovném připojení je vhodné si uživatelské jméno a popřípadě i heslo uložit na lokální disk. V tomto případě nás QGIS upozorní, že ukládáme přihlašovací údaje do nešifrovaného souboru.



Nastavení připojení k databázi nejprve otestujeme a poté potvrdíme.

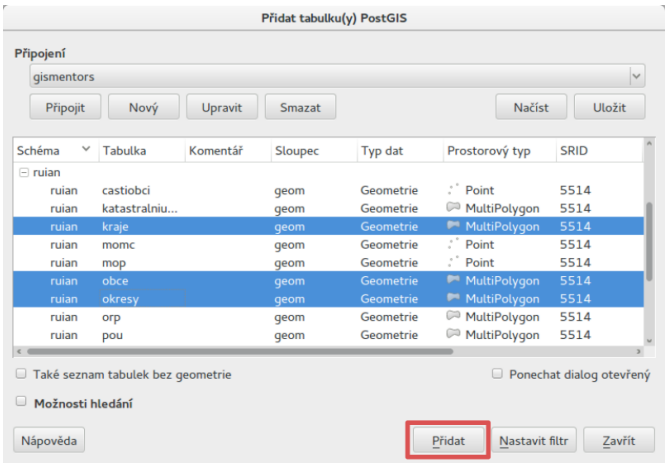
Poznámka pro pokročilé

Připojení k databázi z příkazové řádky

```
psql gismentors -U skoleni -W -h training.gismentors.eu
```

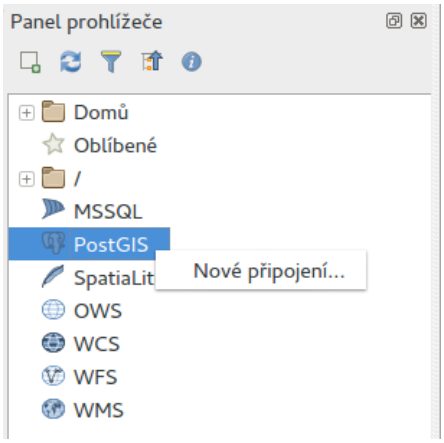
Následně se již můžeme k databázi *připojit*

a vybrat vektorové vrstvy, které chceme z geodatabáze *načíst*.

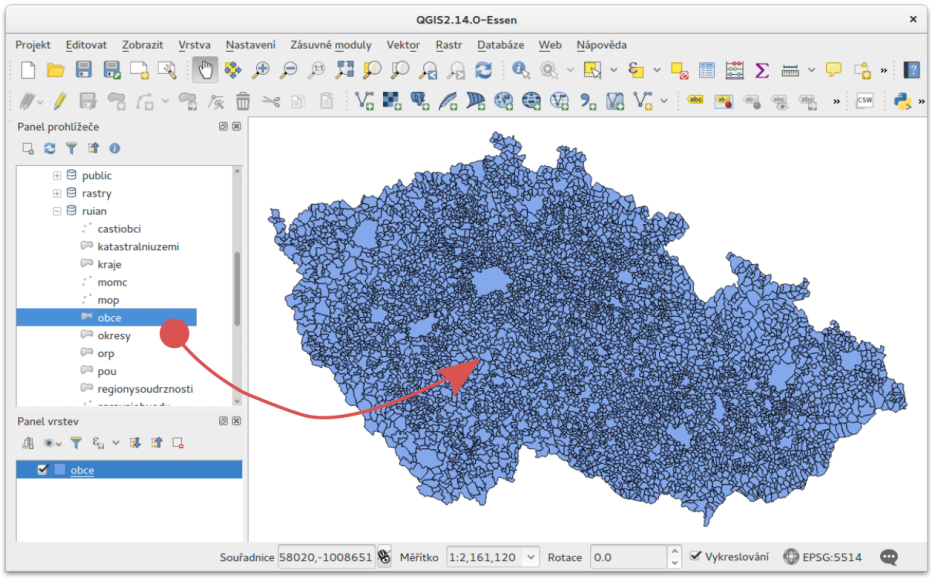


2.1.1 Datový prohlížeč

Připojení k databázi PostGIS je možné definovat i v rámci *datového prohlížeče*.

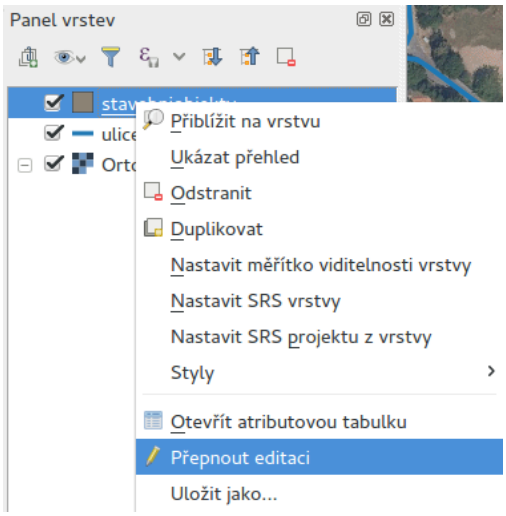


Po připojení k databázi vybranou vektorovou vrstvu jednoduše přetáhneme z datového prohlížeče do mapového okna.



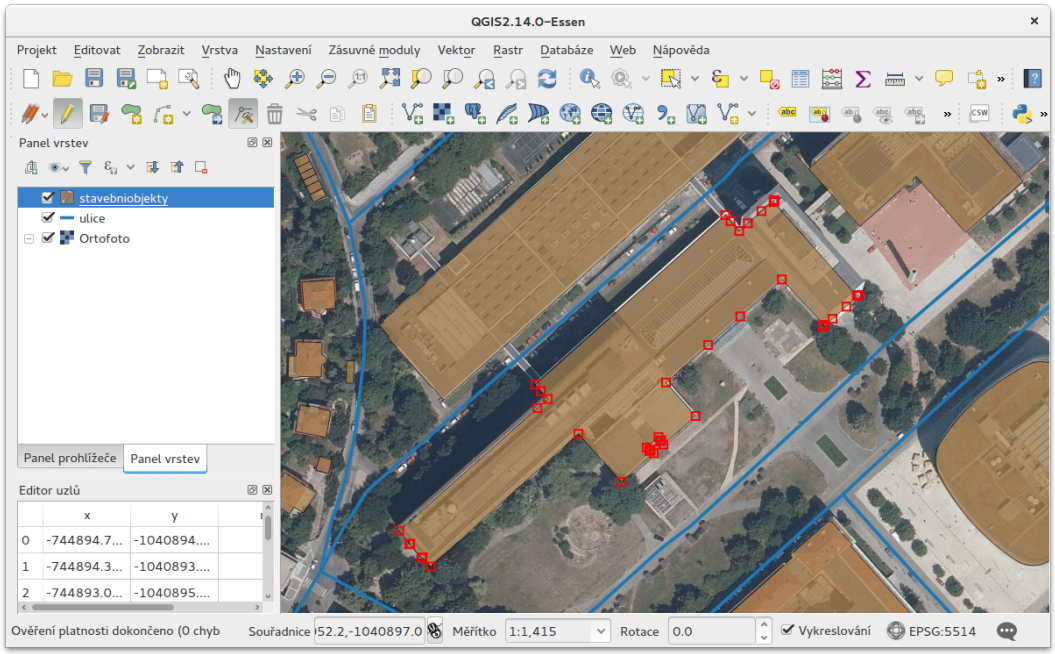
2.2 Editujeme vektorová data

QGIS umožňuje editaci různých formátů vektorových dat včetně dat uložených v geodatabázi PostGIS. Přepnout danou vektorovou vrstvu do *editačního módu* je možné z kontextového menu

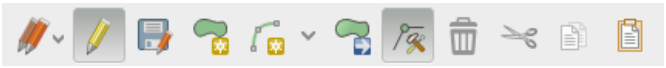


anebo *nástrojové lišty* QGISu  .

Po přepnutí do editačního módu se vektorová vrstva zobrazí včetně lomových bodů (červené křížky).



Editační nástrojová lišta QGISu umožňuje



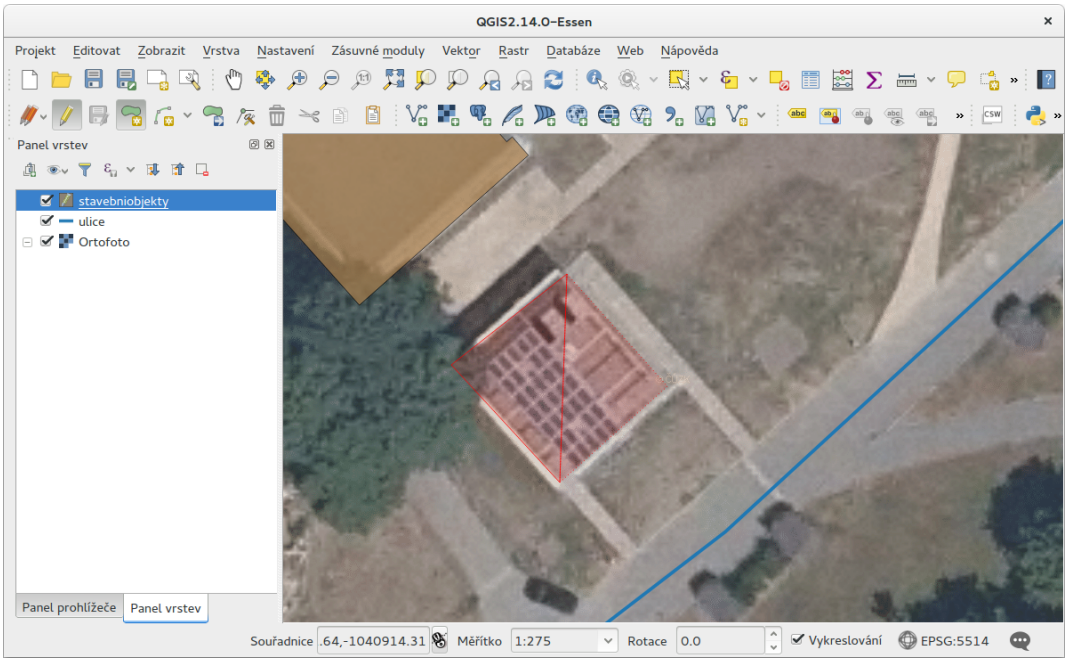
	přidávat nové prvky
	přesunovat existující prvky
	modifikovat uzly (přidávat, mazat a přesunovat)
	smazat vybrané prvky
	vyjmout vybrané prvky
	kopírovat vybrané prvky
	vložit prvky

2.2.1 Příklad přidání nového prvku

Z nástrojové lišty vybereme nástroj pro *přidávání nového prvku*  .

Lomové body nového prvku volíme stisknutím levého tlačítka myši. Poslední uložený lomový bod můžeme vrátit zpět pomocí klávesy `Backspace`.

Editaci prvku ukončíme stisknutím pravého tlačítka myši. Poté se objeví dialog pro zadání atributů nově přidaného prvku.



Tip: Více k tématu editace ve školení QGIS pro začátečníky.

KAPITOLA 3

Jazyk SQL

Jazyk **SQL** je nástroj pro komunikaci uživatele s relační databází. Oproti programovacím jazykům je jednodušší a bližší gramatice mluvené řeči. Je standardizován jako SQL ANSI. V jazyce SQL vytváříme tzv. *dotazy*.

SQL dotazy dělíme na dva základní typy: dotazy pro manipulaci s daty **DML** (data manipulation language) a **DDL** (data definition language) tedy dotazy pro definici dat. DML slouží pro manipulaci se záznamy v tabulkách, tedy pro vyptání dat, mazání záznamů, vyprazdňování tabulek, vkládání a aktualizování záznamů. DDL naopak slouží pro definici databázových struktur. Pro tvorbu databází, tabulek, indexů, pohledů, funkcí, triggerů atd. Dále rozlišujeme **DCL** a **TCL**, tedy data control language a transaction control language. První z nich slouží k nastavení přístupových práv (příkazy GRANT a REVOKE), druhý pak k práci s transakcemi.

Poznámka pro pokročilé

Kromě jazyka SQL můžeme psát v PostgreSQL funkce i v dalších jazycích. Mimo jiné se jedná o Perl, Python, R, JavaScript a další. Zejména však v PL/PgSQL, procedurálním jazyku PostgreSQL svou syntaxí podobného jazyku používanému v databázích Oracle.

3.1 Syntax

Základní kostra jazyka SQL vypadá zhruba následovně:

```
PROVEĎ
S ČÍM
ZA JAKÝCH PODMÍNEK
```

Pro výběr dat z tabulky tedy:

```
VYBER
seznam položek
Z tabulky
PRO KTERÉ PLATÍ
podmínka;
```

Poznámka: SQL dotazy v PostgreSQL zakončujeme středníkem.

3.2 DML

3.2.1 SELECT

Dotaz, kterým vybíráme data z databáze, uvozuje příkaz SELECT následovaný výčtem sloupců požadovaného výstupu. Výčet sloupců může být nahrazen $\star$ pro výběr všech sloupců. Pokud předřadíme výčtu sloupců DISTINCT bude dotaz vracet pouze unikátní kombinace hodnot. Klauzule FROM uvozuje výčet tabulek, ze kterých budeme vybírat a které mohou (ale nemusí) být propojeny klauzulí JOIN. Následovat může výčet podmínek uvedený klauzulí WHERE. Podmínky můžeme řetězit booleovskou logikou pomocí AND, OR, případně vylučovat pomocí NOT.

Nakonec můžeme použít GROUP BY pro sdružování při agregacích, ORDER BY pro sezaření záznamů či případně LIMIT a OFFSET pro omezení řádků výstupu, eventuálně další, méně obvyklé klauzule.

Jak to funguje v praxi?

Dejme tomu, že chcete zjistit, které muchomůrky jsou vhodné k jídlu. Přijdete do knihovny a zeptáte se:

```
Dobrý den, slečno, prosím Vás,
podívala byste se mi do Smotlachova atlasu hub a
zjistila,
které muchomůrky jsou jedlé?
```

Slečna půjde, vytáhne z regálu „Smotlachu“, podívá se do rejstříku a najde všechny muchomůrky, každou nalistuje a zjistí, které jsou jedlé. Ty pro Vás vypíše.

V relační databázi by to vypadalo nějak takto.

Máme **tabulku** nazvanou *smotlacha_atlas_hub*. Vypadá nějak takto:

rod	druh	rod_lat	druh_lat	popis	foto	jedla	vyskyt_lokalita	vyskyt_od	vyskyt_do
muchomůrka	růžovka	amanita	rubescens	…	ruzovka.jpg	true	MULTIPOLYGON(((…	1.6.	31.10.

SQL dotaz potom bude vypadat následovně:

```
SELECT
   rod
   , druh
   , foto
FROM smotlacha_atlas_hub
WHERE
   rod = 'muchomůrka'
   AND jedla = true;
```

V překladu do češtiny by dotaz mohl znít:

```
VYBER
   seznam požadovaných údajů
Z tabulky
[PRO KTERÉ PLATÍ
   podmínka]
```

3.2.2 JOIN

Rozlišujeme dva typy příkazu JOIN, tj. spojení tabulek: INNER JOIN a OUTER JOIN.

INNER JOIN vrátí pouze takové záznamy, kde došlo k nalezení potřebné hodnoty v obou tabulkách. Naproti tomu OUTER JOIN vrací pro jednu, případně obě tabulky všechny záznamy. OUTER JOIN dělíme na LEFT JOIN, RIGHT JOIN a FULL JOIN. LEFT a RIGHT JOIN vrací všechny záznamy z levé nebo pravé tabulky. FULL JOIN vrátí všechny záznamy z obou tabulek. Speciální situací je CROSS JOIN, který vrací kartézský součin záznamů v obou tabulkách.

Záznamy obvykle párujeme pomocí klauzule ON, za kterou následují podmínky propojení podobně jako za klauzulí WHERE. Alternativou je použití klauzule USING, kde je uveden název sloupce, který musí být v obou tabulkách. Další možností je NATURAL JOIN, který použije stejně pojmenované sloupce. Ten však nedoporučeme příliš používat, zvláště v databázích s proměnlivou strukturou.

```
-- vyber "rod druh", "lokalita", "vyskyt"
SELECT houby.rod || ' ' || houby.druh, lokalita.nazev, houby.vyskyt
-- z tabulky houby
FROM houby
-- spoj podle sloupečků s id houby
JOIN lokalita ON houby.id = lokalita.houby_id
-- ale pouze tam, kde lokalita je v oblasti "Vysočina"
WHERE ST_Intersects(
   (
      SELECT geom FROM oblasti WHERE nazev = 'Vysočina'
   )
   , lokalita.geom)
-- a pouze tam, kde výsky je "od"
AND houby.vyskyt @> '2015-07-15'::timestamp;

SELECT houby.rod || ' ' || houby.druh
FROM houby
JOIN r_recept ON r_recept.houby_id = houby.id
```

```
JOIN recept ON recept.id = r_recept.recept_id
WHERE recept.nazev = 'smaženice';
```

3.2.3 UPDATE

UPDATE slouží k aktualizování hodnot vybraných sloupců. Používá se klauzule WHERE a výrazy. Také je možno použít klauzuli FROM a aktualizovat tabulku hodnotami z jiných tabulek.

Příklad nastavení výskytu od 1.června pro všechny houhy z rodu „amanita“:

```
UPDATE smotlacha_atlas_hub SET vyskyt_od = '1.6.' WHERE rod_lat = 'amanita';
```

3.2.4 DELETE

DELETE slouží k mazání vybraných záznamů z tabulek.

Příklad odstranění všech jedlých hub z tabulky:

```
DELETE smotlacha_atlas_hub WHERE jedla = true;
```

3.2.5 TRUNCATE

TRUNCATE slouží k okamžitému vyprázdnění celé tabulky. Je rychlejší, než použití DELETE bez podmínek.

```
TRUNCATE smotlacha_atlas_hub;
```

3.2.6 Množinové operace

Množinové operace pracují s výsledky více poddotazů. Jedná se o UNION, UNION ALL, EXCEPT a INTERSECT.

UNION vrací sjednocení záznamů z obou dotazů. Záznamy, které jsou výsledkem (tvz. *recordset*) obou dotazů, jsou po sjednocení obsaženy pouze jednou. Naproti tomu UNION ALL vrátí všechny záznamy, výsledkem sjednocení je tedy součet záznamů z obou recordsetů.

Poznámka pro pokročilé

Pokud víme, že záznamy se mezi dotazy neduplikují, je lepší použít UNION ALL. Provádění pak bude efektivnější, protože si ušetříme porovnávání obou výstupních recordsetů.

EXCEPT vrací rozdíl, tedy pouze takové záznamy, které se vyskytují pouze v prvním recordsetu. INTERSECT vrací jejich průnik. Tedy záznamy, které se vyskytují v obou recordsetech.

3.2.7 Poddotazy

V rámci dotazu můžeme dotazovat další *vnořené* dotazy uzavřené do závorek.

```
SELECT recepty.* FROM
(
SELECT DISTINCT recept_id FROM r_recept WHERE houby_id IN
   (
      SELECT * FROM houby WHERE rod = 'bedla'
   )
) recepty_na_bedly
JOIN recepty ON recepty.id = recepty_na_bedly.recept_id;
```

3.3 DDL

CREATE a DROP jsou základní příkazy z *Data Definition Language*. Pomocí nich vytváříme tabulky, pohledy, omezení, funkce, typy a další.

CREATE TABLE

CREATE VIEW

CREATE FUNCTION

CREATE LANGUAGE

...

3.4 A co prostorová databáze?

Dejme tomu, že nás zajímají jen ty houby, které rostou v okruhu třiceti kilometrů od Pece pod Sněžkou, kde hodláme strávit dovolenou.

V takovém případě slečna musí porovnat místo výskytu s vámi zadanou lokalitou.

Poznámka pro pokročilé

Je zjevné, že k požadovanému výsledku se může slečna dobrat různými, různě efektivními způsoby. Postup, kterým bude pracovat se nazývá *prováděcí plán* (query plan). K volbě ideálního způsobu slouží statistiky, které si databáze ukládá a které jsou aktualizovány po každém dotazu.

Dotaz do SQL může potom vypadat následovně:

```
SELECT
   rod
   , druh
   , foto
FROM smotlacha_atlas_hub
WHERE
```

```
  rod = 'muchomůrka'
  AND jedla = true
  AND ST_Distance(vyskyt_lokalita,
  'SRID=5514;POINT(-641455 -987918)'::geometry) < 3e4;
```

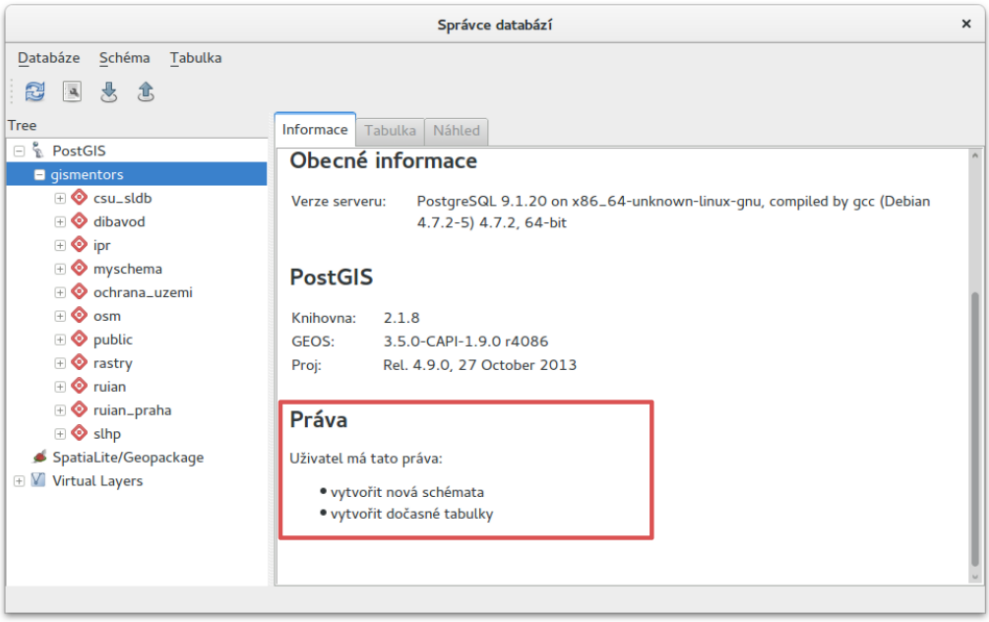
KAPITOLA 4

Prostorové dotazy

4.1 Připojujeme se do databáze z QGIS

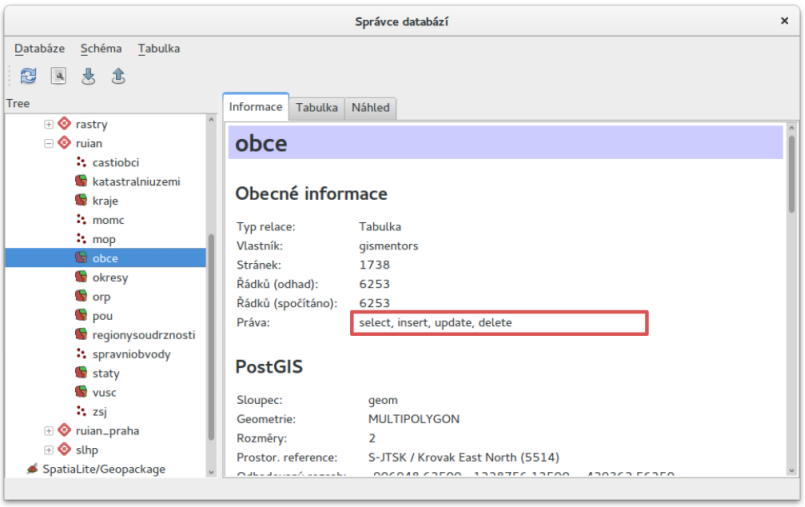
Přístup do databáze umožnuje zásuvný modul QGISu **`DB Manager`** (Správce databází). DB Manager spustíme z menu aplikace QGIS *Databáze* → *Správce databází* → *Správce databází*.

V dialogu vybereme databázi školení „gismentors“.



Obr. 4.1: Uživatel má v tomto případě právo v databázi vytvářet vlastní schémata a dočasné tabulky.

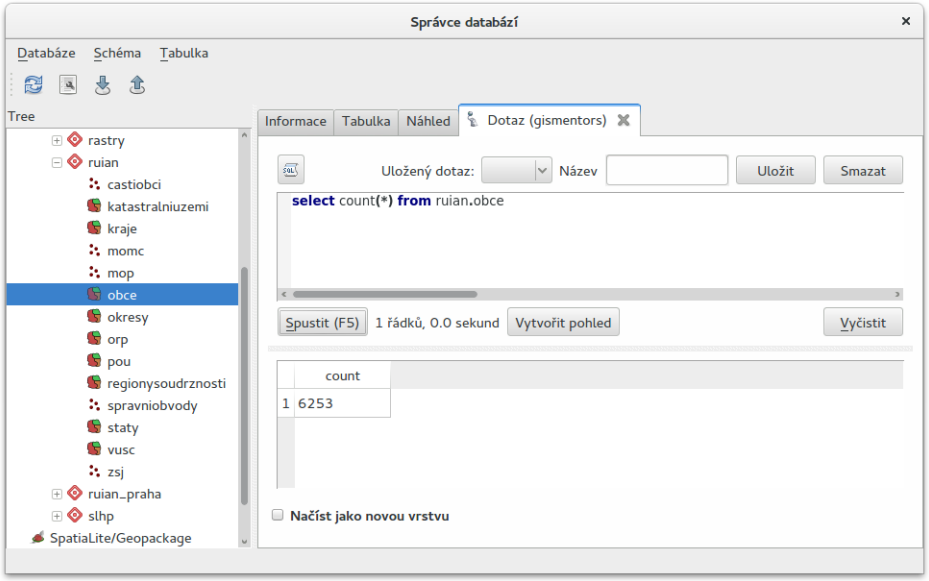
Můžeme procházet metadata jednotlivých vrstev uložených v geodatabázi.



Obr. 4.2: Uživatel má v tomto případě pro vrstvu *obce* ve schématu *ruian* veškerá práva a může ji modifikovat.

4.1.1 Provádíme SQL dotazy

Otevřeme dialog SQL okna  , které nám umožní provádět jednoduché *SQL dotazy* přímo v prostředí aplikace QGIS.



Obr. 4.3: Příklad určení počtu obcí v ČR.

Tip: Pokročilejší uživatelé ocení spíše konzolový nástroj **`psql`**. Více k tomuto tématu ve školení PostGIS pro pokročilé.

Vytváříme novou vrstvu jako výsledek prostorového dotazu

Na základě prostorového dotazu můžeme pomocí dialogu *správce databází* vytvářet nové datové vrstvy.

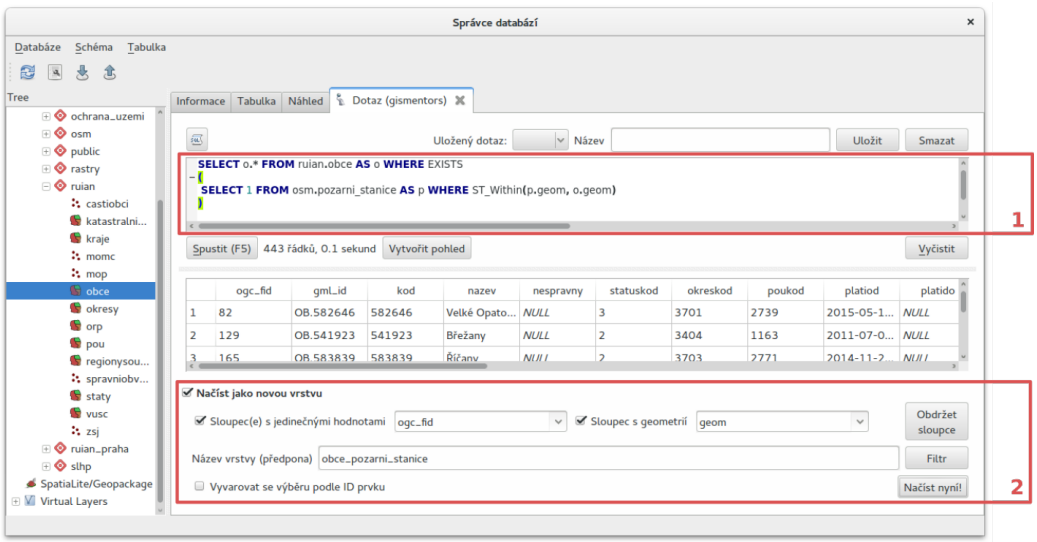
V nasledujícím příkladě vybereme (1) obce (ruian.obce_polygon), které obsahují alespoň jednu pořární stanici (osm.pozarni_stanice). Výsledek zobrazíme v QGISu jako novou vrstvu obce_pozarni_stanice (2).

Poznámka:

```
SELECT o.* FROM ruian.obce AS o JOIN osm.pozarni_stanice AS p
 ON ST_Within(p.geom, o.geom);
```

Dotaz vracím obce, ve kterých je více než jedna požární stanice, jako duplicitní. Správně by tento dotaz mohl vypadat např. následovně:

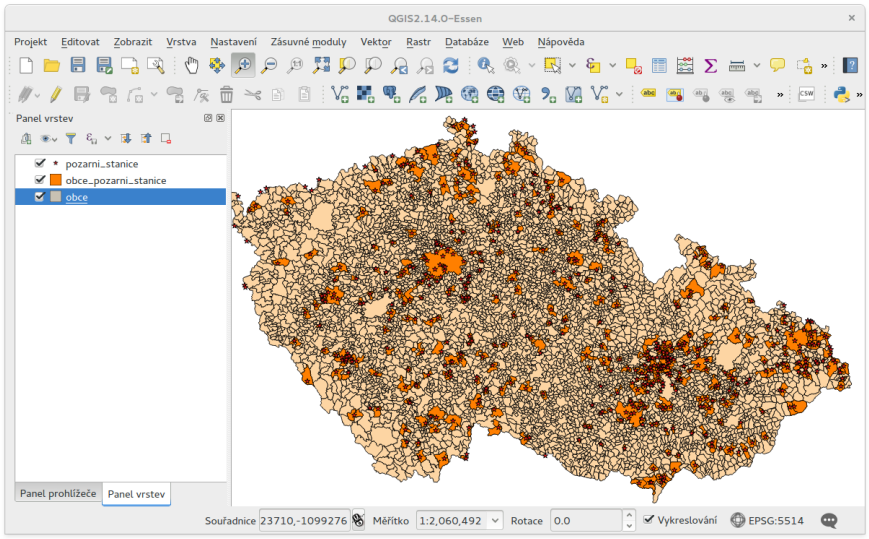
```
SELECT o.* FROM ruian.obce AS o WHERE EXISTS
(
 SELECT 1 FROM osm.pozarni_stanice AS p WHERE ST_Within(p.geom, o.geom)
);
```



Poznámka: Alternativně můžete novou vrsvu vytvořit v databázi rovnou jako novou tabulku anebo pohled a zobrazit v QGISu standardní cestou.

```
-- nejprve vytvoříme vlastní schéma
CREATE SCHEMA uzivatel;

CREATE VIEW uzivatel.obce_pozarni_stanice AS
SELECT o.* FROM ruian.obce_polygon AS o WHERE EXISTS
(
 SELECT 1 FROM osm.pozarni_stanice AS p WHERE ST_Within(p.geom, o.geom)
);
```

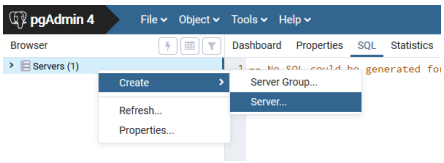


Obr. 4.4: Výsledek prostorového dotazu.

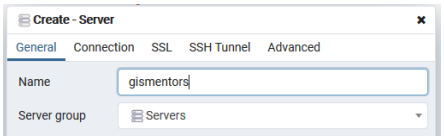
4.2 Přístup z PgAdmin

SQL dotazy můžeme provádět v grafické uživatelském prostředí PgAdmin. V následujícím textu předpokládáme verzi PgAdmin 4.

Poznámka: Pokud používáte QGIS, tak Vám PgAdmin nepřinese nic nového, spíše naopak. PgAdmin není GIS aplikace (od verze 3.3 PgAdmin4 nabízí alespoň jednoduchou prohlížečku geografických dat). Neumožní Vám zobrazit výsledky prostorových dotazů v mapovém okně podobně jako QGIS. Jde o grafické uživatelské rozhraní pro přístup k databázi PostgreSQL, nic víc. Navíc nepodporuje našeptávání a další užitečné funkce. Pro efektivní práci s databází se nejvíce hodí konzolový klient **`psql`**, více na školení PostGIS pro pokročilé.

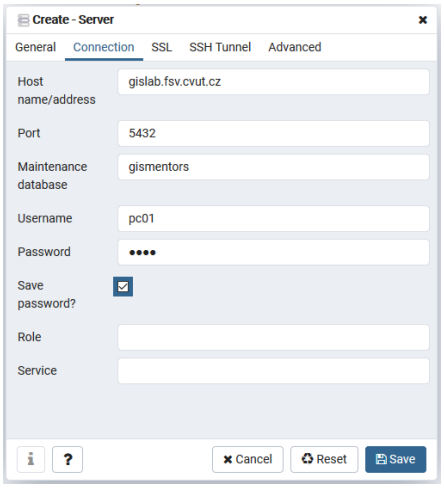


Obr. 4.5: Přidáme nové spojení.

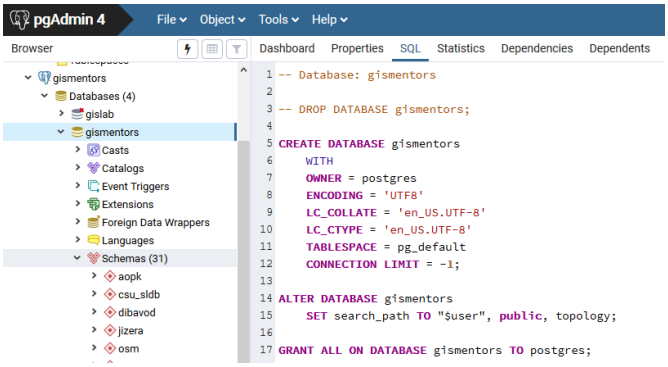


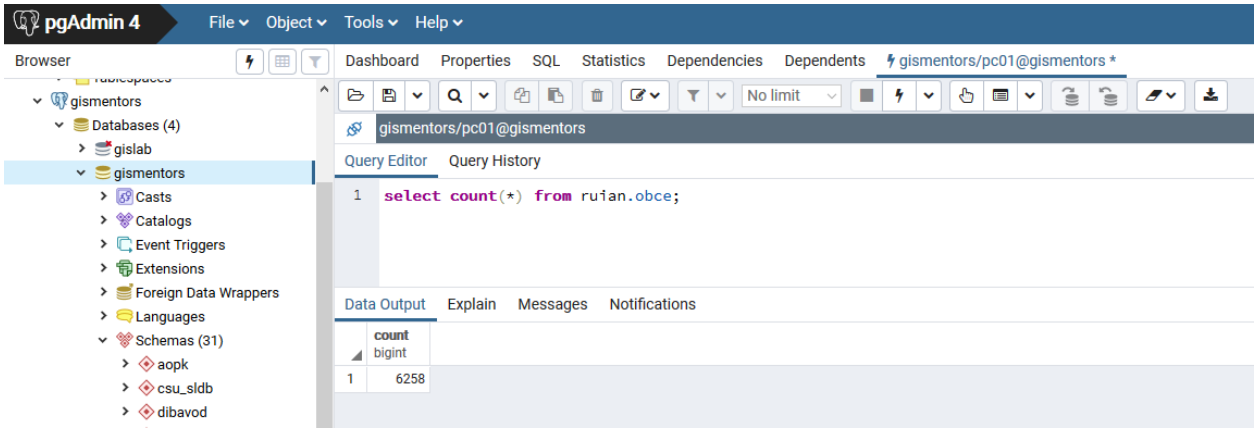
Do seznamu se přidá nová položka.

Z menu *Tools* → *Query Tool* otevřeme nástroj, který nám umožní provádět SQL dotazy.



Obr. 4.6: V následujícím dialogu zadáme název připojení a především parametry připojení k databázi.





Obr. 4.7: Příklad určení počtu obcí v ČR.

KAPITOLA 5

Import a export dat z databáze

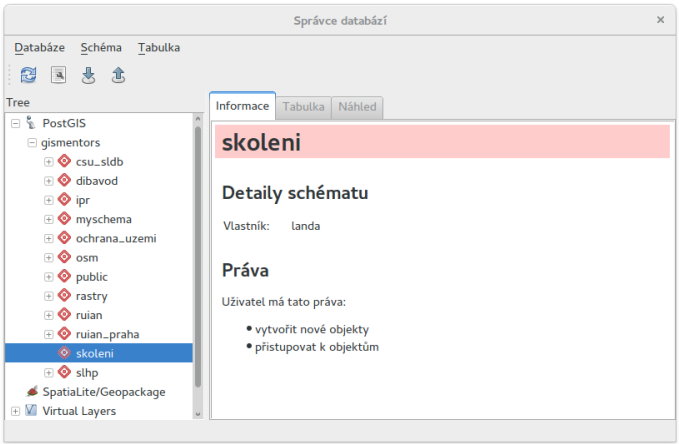
5.1 Nahráváme vlastní data do databáze

5.1.1 Správce databází

Předpokládáme, že každý uživatel pracuje ve vlastní databázovém schématu. Toto schéma vytvoříme pomocí *správce databází* v QGISu.

Vytvoření databázového schématu

V našem případě uložíme vektorová data do *vlastního schématu*, nejprve toto schéma vytvoříme *Schéma* → *Vytvořit schéma*.

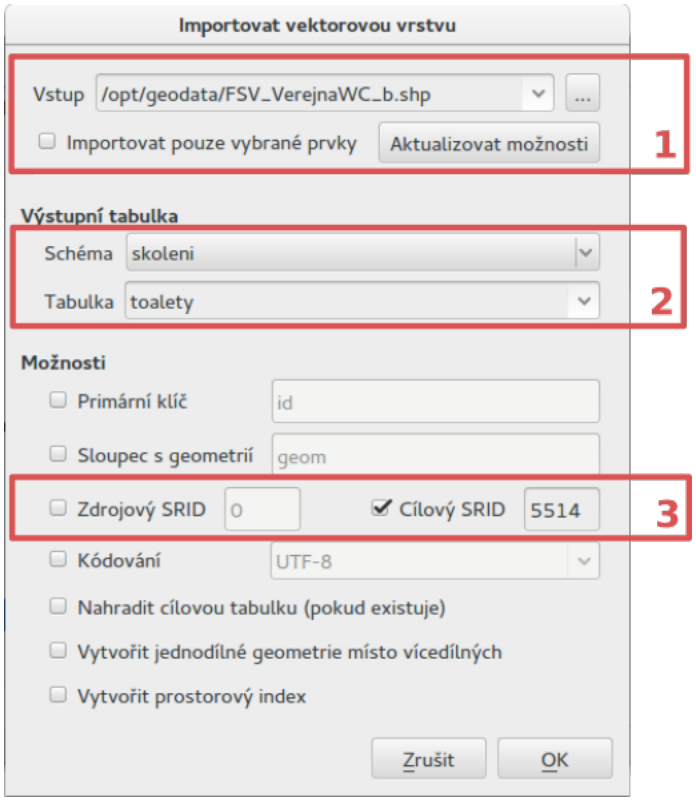


Obr. 5.1: V nově vytvořeném schématu již má uživatel „skoleni“ právo zápisu.

Import dat

Nahrání geodat do databáze PostGIS umožňuje v QGISu samotný *správce databází*. Soubor s geodaty anebo načtenou vrstvu v QGISu naimportujeme z menu *Tabulka* → *Importovat vrstvu/soubor* anebo z nástrojové lišty správce databází  .

V dialogu vybereme soubor pro import do geodatabáze (1). Dále můžeme změnit cílové schéma a název výsledné tabulky v databázi (2). Dialog nabízí další možnosti včetně transformace do jiného souřadnicového systému (pokud je zadán současně zdrojový a cílový souřadnicivový systém) anebo prosté vynucení cílového souřadnicového systému (3).



Poznámka: Ve níže uvedeném případě importujeme vrstvu veřejných toalet z otevřené datové sady IPR.

Naimportovaná vrstva z geodatabáze PostGIS se nezobrazí automaticky, musíte ji do mapového okna *přidat manuálně*.

5.1.2 Další možnosti

Spit

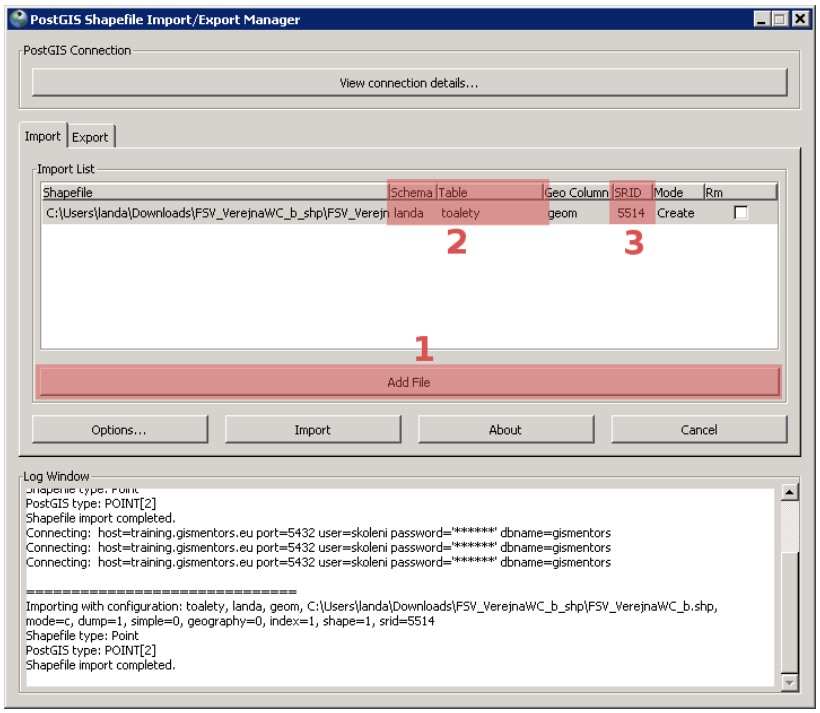
Import vektorových dat ve formátu *Esri Shapefile* umožňuje také zásuvný modul *Spit (Shapefile import)* dostupný z menu aplikace QGIS *Spit (Shapefile import)* → *Import Shapefile do PostgreSQL*.

Varování: Zásuvný modul Spit není od verze QGIS 2.14 podporován a nijak udržován. Pro import dat se doporučuje používat *správce databází*.

pgAdmin

Vektorová data ve formátu *Esri Shapefile* lze do databáze PostGIS naimportovat pomocí zásuvného modulu *PostGIS Shapefile and DBF loader* aplikace *PgAdmin* *Zásuvné moduly* → *PostGIS Shapefile and DBF loader*.

V dialogu pro import definujeme vstupní soubor ve formátu Esri Shapefile (1), cílové databázové schéma a cílovou tabulku (2) a případně i souřadnicový systém (3).



5.1.3 Pro pokročilé uživatele

Tip: Více k tomuto tématu ve školení PostGIS pro pokročilé.

shp2pgsql

shp2pgsql je konzolový nástroj, který umožňuje import vektorových dat ve formátu *Esri Shapefile* do geodatabáze PostGIS. Tento nástroj je součástí instalace PostGIS.

Import dat do databáze pomocí shp2pgsql z příkazové řádky

Nejprve vytvoříme SQL dávku

```
shp2pgsql -s 5514 FSV_VerejnaWC_b.shp skoleni.toalety > wc.sql
```

- `-s` definuje souřadnicový systém (v tomto případě EPSG:5514),
- `FSV_VerejnaWC_b.shp` je název vstupního souboru ve formátu Esri Shapefile,
- `landa.toalety` je název výstupního databázového schématu a tabulky (oddělené tečkou),
- `> wc.sql` dávka je uložena do souboru `wc.sql`.

Vytvořenou SQL dávku nahrajeme do databáze *gismentors* přes nástroj **`psql`** a jeho parametr `-f`:

```
psql gismentors -U skoleni -W -h training.gismentors.eu -f wc.sql
```

ogr2ogr

ogr2ogr je konzolový nástroj knihovny GDAL umožňující konverzi mezi datovými formáty podporovanými touto knihovnou.

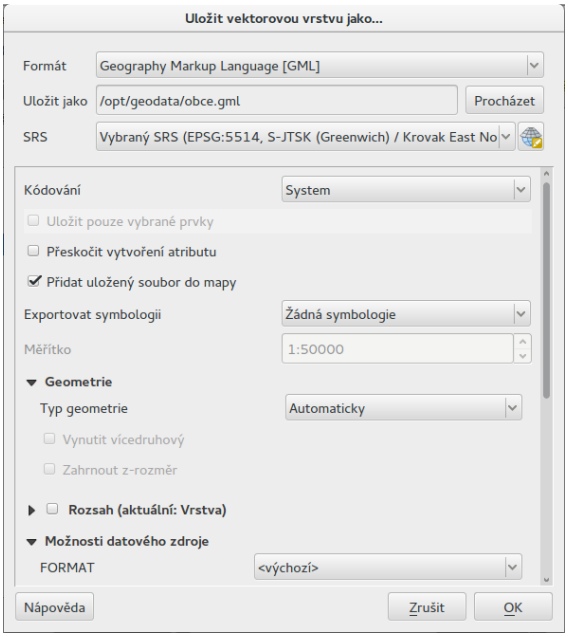
Import dat do databáze pomocí ogr2ogr z příkazové řádky

```
ogr2ogr -f PostgreSQL \
PG:"dbname=gismentors host=training.gismentors.eu user=skoleni password=XXX \
active_schema=skoleni" \
FSV_VerejnaWC_b.shp \
-a_srs EPSG:5514
```

5.2 Export dat z databáze

Data můžeme exportovat z databáze v prostředí QGIS naprosto stejně jako u jiných formátů. Načteme si do QGIS vrstvu, kterou si přejeme vyexportovat a z kontextového menu nad vrstvou zvolíme volbu Save As.

V následujícím dialogu zvolíme požadovaný výstupní formát a případně další volby, kterou jsou již závislé na zvoleném formátu.



Obr. 5.2: Příklad exportu vektorových dat z databáze do formátu OGC GML.

5.2.1 Pro pokročilé uživatele

Podobně jako v případě importu dat, lze použít pokročilejší konzolové nástroje. Ty je možné volat ve skriptech při automatizaci apod. Ukážeme si použití nástroje **pgsql2shp**, který umožňuje export dat do formátu Esri Shapefile a **ogr2ogr** knihovny GDAL.

Tip: Více k tomuto tématu ve školení PostGIS pro pokročilé.

pgsql2shp

PostGIS kromě nástroje pro import dat ve formátu Esri Shapefile **shp2pgsql** nabízí obdobný nástroj pro export dat **pgsql2shp**.

Export do formátu Esri Shapefile pomocí pgsql2shp z příkazové řádky

V níže uvedeném příkladě vyexportujeme tabulku obce ze schéma *ruain* do souboru *obce.shp*.

```
pgsql2shp -h training.gismentors.eu -u skoleni -P XXX -f obce gismentors \
ruian.obce
```

ogr2ogr

ogr2ogr slouží obecně ke konverzi dat, lze jej tedy použít jak pro import tak export dat.

Export do formátu Esri Shapefile pomocí ogr2ogr z příkazové řádky

```
ogr2ogr -f 'ESRI Shapefile' \
-lco 'ENCODING=UTF-8' \
obce.shp \
PG:"dbname=gismentors host=training.gismentors.eu user=skoleni password=XXX" \
ruian.obce
```

Na rozdíl od nástroje **pgsql2shp** umožňuje **ogr2ogr** export nejen do formátu Esri Shapefile, ale do celé řady formátů, které knihovna GDAL porpodruje v režimu zápisu.

Export do formátu GML pomocí ogr2ogr z příkazové řádky

```
ogr2ogr -f 'GML' \
obce.gml \
PG:"dbname=gismentors host=training.gismentors.eu user=skoleni password=XXX" \
ruian.obce_polygon
```

KAPITOLA 6

Přehled vybraných prostorových funkcí a operátorů

6.1 Operátory

Mezi nejpoužívanější operátory patří:

- `&&` - překryv minimálních ohraničujících obdélníků
- `<#>` - vzdálenost minimálních ohraničujících obdélníků
- `<->` - vzdálenost centroidů

Poznámka: Více informací v dokumentaci PostGIS.

6.2 Konstruktory

Kostruktory jsou funkce vytvářející v PostGISu nové geometrické objekty.

ST_GeomFromText geometrie z WKT.

ST_GeomFromGML, ST_GeomFromWKB, ST_Point, ...

K vytvoření geometrie lze použít kromě funkcí PostGISu i přetypování z WKT:

```
SELECT
'SRID=5514; POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry(POLYGON, 5514);
```

Poznámka: Více informací v dokumentaci PostGIS.

6.3 Výstupy

Funkce umožňující převod geometrie do jiné reprezentace.

ST_AsText, ST_AsGML, ST_AsSVG, ...

6.4 Rozměr geometrie

ST_Area plocha.

ST_Perimeter obvod.

ST_Length délka.

6.5 Další operace nad geometrií

ST_ExteriorRing obvodová hranice.

ST_Dump rozdělí multipart geometrii.

ST_Polygonize zaplochování.

ST_ConvexHull konvexní obal.

ST_Translate posun.

ST_Buffer obalová zóna.

ST_SetSRID nastaví SRID (ID souřadnicového systému)

...

6.6 Vzájemná poloha dvou geometrií

ST_Relate devítiprvková matice.

ST_Intersects existuje průnik (i jeden bod).

```
SELECT
ST_Intersects(
   'POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry
   , ST_Translate('POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry,1,1)
);
```

ST_Disjoint, ST_Overlaps, ST_Crosses, ST_Within, ST_DWithin, ST_Touches

6.7 Geometrické operace

ST_Intersection průnik.

ST_Difference rozdíl.

ST_SymDifference symetrický rozdíl.

6.8 Agregace

ST_Union sjednocení.

KAPITOLA 7

Praktické ukázky

7.1 Jak vypsat všechny tabulky s geometrií

Nejdříve si ukážeme, jak rychle zjistit, které tabulky v databázi obsahují prostorová data.

Tuto informaci získáme z pohledu geometry_columns. Ten zobrazuje data ze systémových tabulek (data o typech a omezeních) a přehledně je zobrazuje.

Poznámka: Ve verzích PostGIS 1.x byl geometry_columns definován jako tabulka a nikoliv jako pohled.

Jeho struktura je následující:

Sloupec	Typ	Uložení
f_table_catalog	character varying(256)	extended
f_table_schema	character varying(256)	extended
f_table_name	character varying(256)	extended
f_geometry_column	character varying(256)	extended
coord_dimension	integer	plain
srid	integer	plain
type	character varying(30)	extended

Poznámka pro pokročilé

PostGIS definuje pohled geometry_columns následovně:

```
 SELECT current_database()::character varying(256) AS f_table_catalog,
  n.nspname::character varying(256) AS f_table_schema,
  c.relname::character varying(256) AS f_table_name,
  a.attname::character varying(256) AS f_geometry_column,
  COALESCE(NULLIF(postgis_typmod_dims(a.atttypmod), 2)
     , postgis_constraint_dims(n.nspname::text, c.relname::text, a.
↪attname::text)
     , 2) AS coord_dimension,
  COALESCE(NULLIF(postgis_typmod_srid(a.atttypmod), 0)
     , postgis_constraint_srid(n.nspname::text, c.relname::text, a.
↪attname::text)
     , 0) AS srid,
  replace(replace(COALESCE(NULLIF(upper(postgis_typmod_type(a.atttypmod)),
     'GEOMETRY'::text)
     , postgis_constraint_type(n.nspname::text, c.relname::text,
       a.attname::text)::text
     , 'GEOMETRY'::text), 'ZM'::text, ''::text), 'Z'::text,
       ''::text)::character varying(30) AS type
 FROM pg_class c,
  pg_attribute a,
  pg_type t,
  pg_namespace n
WHERE t.typname = 'geometry'::name
AND a.attisdropped = false AND a.atttypid = t.oid AND
 a.attrelid = c.oid AND c.relnamespace = n.oid
AND (c.relkind = 'r'::"char" OR c.relkind = 'v'::"char" OR
 c.relkind = 'm'::"char" OR c.relkind = 'f'::"char")
AND NOT pg_is_other_temp_schema(c.relnamespace)
AND NOT (n.nspname = 'public'::name AND c.relname = 'raster_columns'::name)
AND has_table_privilege(c.oid, 'SELECT'::text);
```

Provedeme jednoduchý dotaz do tohoto pohledu.

```
SELECT
*
FROM geometry_columns
WHERE f_table_schema = 'dibavod';
```

Vybíráme tedy všechny záznamy vztažené k tabulkám ze schématu *dibavod*.

Výsledek může vypadat například takto:

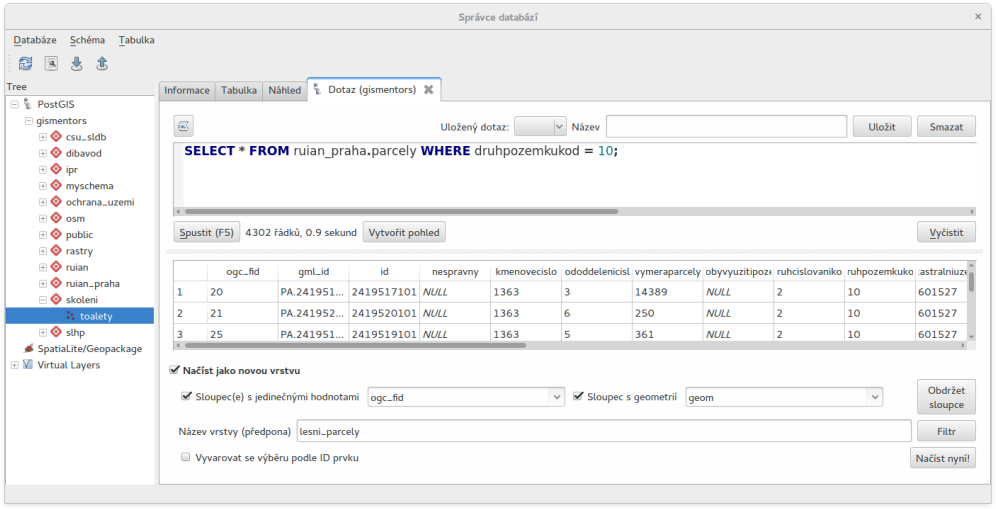
```
f_table_catalog   | gismentors
f_table_schema    | dibavod
f_table_name      | povodi_iii
f_geometry_column | geom
coord_dimension   | 2
srid              | 5514
type              | MULTIPOLYGON
...
```

7.2 Jednoduchý atributový dotaz

1. Vyberte parcely ze schématu *ruian_praha*:
   - s kódem ochrany *26* (pozemek určený k plnění funkcí lesa)
   - s druhem pozemku *10* (les)

Poznámka: Nezapomeneme zkontrolovat, zda je sloupec, který dotazujeme, *oindexován*.

```
SELECT * FROM ruian_praha.parcely WHERE druhpozemkukod = 10;
```



Obr. 7.1: Ukázka SQL dotazu ve správci databází, výsledek dotazu je zobrazen v QGISu jako nová mapová vrstva.

```
SELECT * FROM ruian_praha.parcely WHERE druhpozemkukod = 26;
```

2. Vyberte stavební objekty ze schématu *ruian_praha* vybavené
   - plynem
   - výtahem a obarvěte je podle počtu podlaží

```
SELECT * FROM ruian_praha.stavebniobjekty
WHERE pripojeniplynkod IS NOT NULL;
```

```
SELECT * FROM ruian_praha.stavebniobjekty
WHERE vybavenivytahemkod IS NOT NULL;
```

Obr. 7.2: Budovy v Praze s výtahem obarvené podle počtu podlaží.

7.3 Jednoduchý prostorový dotaz

1. Vypiště název obce a její rozlohu v hektarech

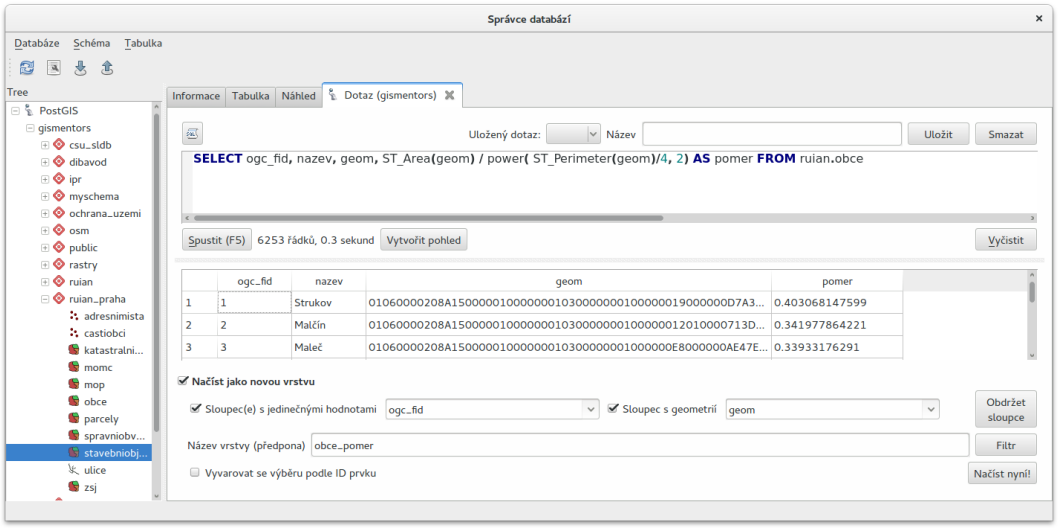
```
SELECT
nazev
, ST_Area(geom)/1e4 AS rozloha
FROM ruian.obce
ORDER BY nazev;
```

2. Zobrazte obce větší než 130 ha

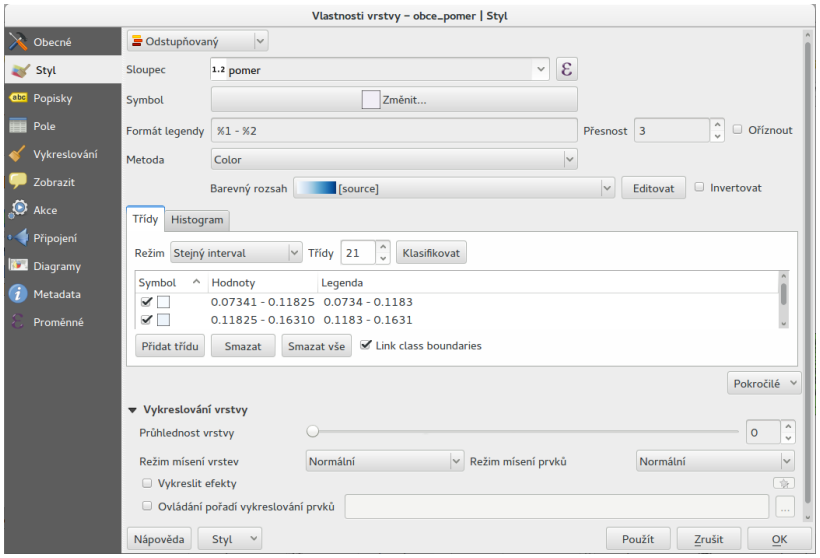
```
SELECT
*
FROM ruian.obce
WHERE ST_Area(geom)/1e4 > 130;
```

3. Nasymbolizujte vrstvu ruian.obce podle poměru rozlohy a čtvrtiny obvodu na druhou.

```
SELECT ogc_fid
, nazev
, geom
, ST_Area(geom) / power( ST_Perimeter(geom)/4, 2) AS pomer
FROM ruian.obce;
```



Obr. 7.3: Výsledek dotazu nahrajeme do QGISu jako novou mapovou vrstvu.



Obr. 7.4: Symbolizaci vrstvy provedeme v QGISu.

7.4 Atributový JOIN

1. Obarvěte obce (ruian.obce)
    - podle počtu obyvatel (csu_sldb.sldb)
    - počtu obyvatel na kilometr čtvereční

```
SELECT
ogc_fid, o.nazev, geom, vse1111/(ST_Area(geom)/1e6) pocet_obyv_na_km
FROM ruian.obce o
JOIN csu_sldb.sldb s ON
s.uzcis = '43' --obce
```

```
AND s.uzkod = o.kod
order by vse1111/(ST_Area(geom)/1e6)
```

2. Obarvěte ORP (ruian.orp)
    - podle zastoupení jasanu (slhp.slhp)

```
SELECT o.ogc_fid
, o.nazev
, o.geom
, s.plocha_proc
FROM ruian.orp o
JOIN slhp.slhp s
ON s.orp_kod = o.kod
WHERE drevina = 'jasan'
```

7.5 Prostorový JOIN

1. Vyberte obce, na jejichž území je požární stanice.

```
SELECT o.nazev
FROM ruian.obce o
JOIN osm.pozarni_stanice p ON ST_Within(p.geom, o.geom)
GROUP BY o.kod, o.nazev;
```

2. Najděte obce, na jejichž území leží více než jedna požární stanice.

```
SELECT o.nazev, count(*)
FROM ruian.obce o
JOIN osm.pozarni_stanice p ON ST_Within(p.geom, o.geom)
GROUP BY o.kod, o.nazev
HAVING count(*) > 1
ORDER BY count(*) DESC;
```

3. Na území které obce leží nejvíce požárních stanic?

```
SELECT o.nazev, count(*)
FROM ruian.obce o
JOIN osm.pozarni_stanice p ON ST_Within(p.geom, o.geom)
GROUP BY o.kod, o.nazev
ORDER BY count(*) DESC
LIMIT 1;
```

4. Vyberte parcely v Praze, které leží na MZCHU.

```
BEGIN;
CREATE TABLE jelen.parcely_mzchu AS
SELECT * FROM ruian_praha.parcely p
WHERE EXISTS (
   SELECT * FROM ochrana_uzemi.maloplosna_uzemi m
```

```
   WHERE ST_Intersects(p.geom, m.geom)
);

ALTER TABLE jelen.parcely_mzchu ADD PRIMARY KEY (ogc_fid);

CREATE INDEX ON jelen.parcely_mzchu USING GIST(geom);

COMMIT;
```

5. Které z nich nemají správně kód způsobu ochrany?
6. Najděte v Praze budovy ohrožené stoletou vodou.

7.6 Buffer

1. Vytvořte obalovou zónu s tloušťkou klesající s řádem kolem vodních toků (dibavod.vodni_toky)

```
--vybere povodi Jizery pomoci rekurze
CREATE TABLE jelen.povodi_jizery AS
WITH RECURSIVE povodi_jizery AS (
        SELECT
        * , 1 rad
        FROM dibavod.vodni_toky
        WHERE tok_id =  110740000100
        UNION ALL
        SELECT
        v.*, j.rad + 1
        FROM dibavod.vodni_toky v
        JOIN povodi_jizery j
        ON j.tok_id = v.tokrec_id
)
SELECT row_number() over() rid, * FROM povodi_jizery;
```

Poznámka pro pokročilé

```
BEGIN;

CREATE TABLE jelen.jize (ogc_fid serial primary key,
 geom geometry(LINESTRING, 5514));

INSERT INTO jelen.jize (geom)
SELECT (ST_Dump(geom)).geom FROM
(
   SELECT ST_Union(geom) geom FROM jelen.povodi_jizery
) uni
;

CREATE INDEX ON jize USING gist(geom);

ALTER TABLE jize ADD rad smallint, ADD parent int;
```

```
UPDATE jize
SET parent = 0, rad = 1 WHERE ogc_fid = 2040;


DO $$
   DECLARE i int;
   BEGIN
      WHILE (SELECT count(*) FROM jize WHERE rad IS NULL) > 0
         LOOP
            UPDATE jize j
            SET rad = r.rad+1, parent = r.ogc_fid
            FROM jize r
            WHERE r.rad IS NOT NULL
            AND j.rad IS NULL
            AND ST_Touches(j.geom, r.geom)
            ;

            RAISE NOTICE '%', count(*) FROM jize WHERE rad IS NULL;



         END LOOP;
   END
   $$;

COMMIT;
```

```
SELECT
rid
, ST_Buffer(geom, 90-(rad * 10)) geom
FROM jelen.povodi_jizery;
```

7.7 Agregace

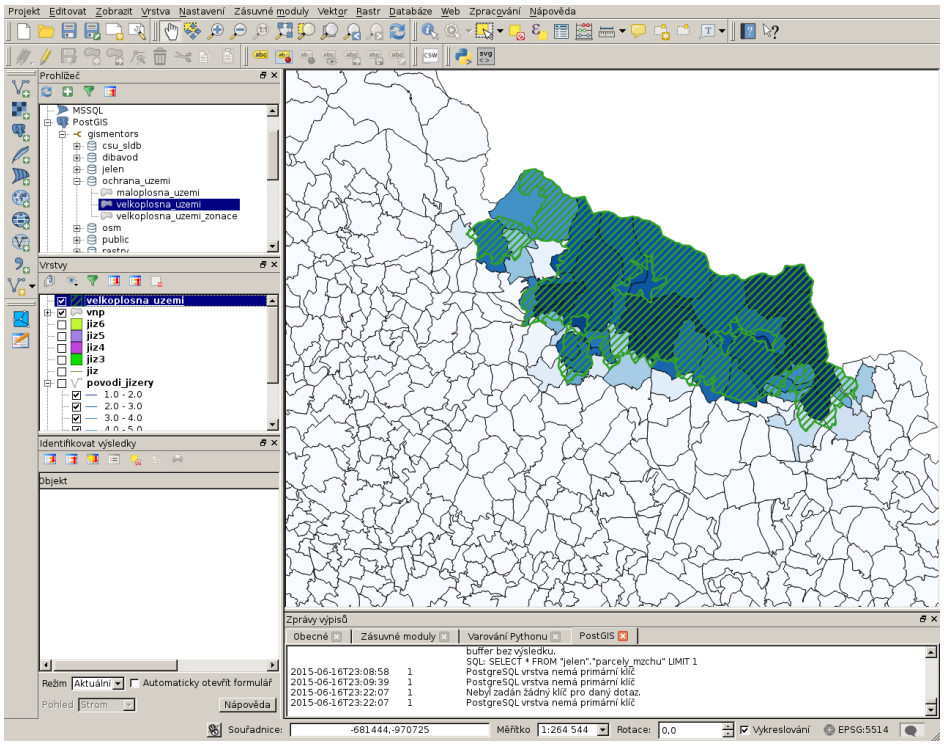
1. Vytvořte mapu POU (pověřené obce) z vrstvy obcí.

```
SELECT
ST_Union(geom) geom
, poukod
FROM ruian.obce
GROUP BY poukod;
```

7.8 Prostorové analýzy

1. Obarvěte katastrální území podle toho, kolik procent území je v NP

```
SELECT
katuze.*
, COALESCE (
        (ST_Area(ST_Intersection(katuze.geom, vzchu.geom))/
        ST_Area(katuze.geom)) * 100
, 0) v_np
FROM
ruian.katastralniuzemi katuze
LEFT JOIN
(
        SELECT
        k.ogc_fid
        , ST_Union(vzchu.geom) geom
        FROM
        ruian.katastralniuzemi k
        JOIN ochrana_uzemi.velkoplosna_uzemi vzchu
        ON vzchu.geom && k.geom
        AND vzchu.kat = 'NP'
        GROUP BY k.ogc_fid
) vzchu
USING(ogc_fid)
```



Obr. 7.5: Vizualizace výsledku v QGISu.

KAPITOLA 8

Poznámky k instalaci a obnově databáze

8.1 GNU/Linux

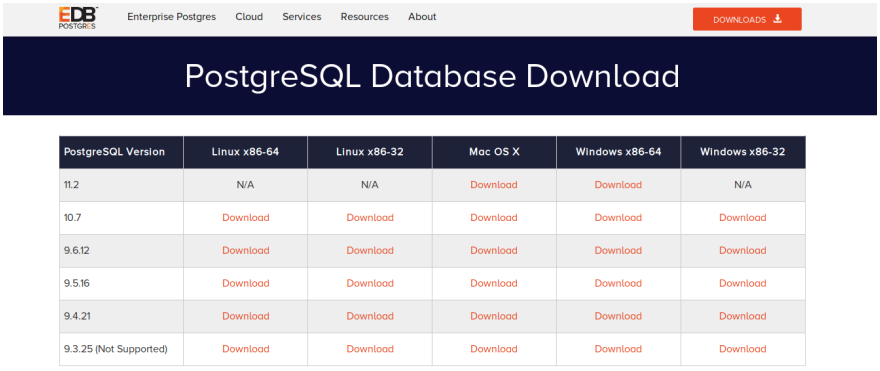
Z distribučního balíčku dané Linuxové distribuce.

8.1.1 Ubuntu / Debian

```
sudo apt install postgis
```

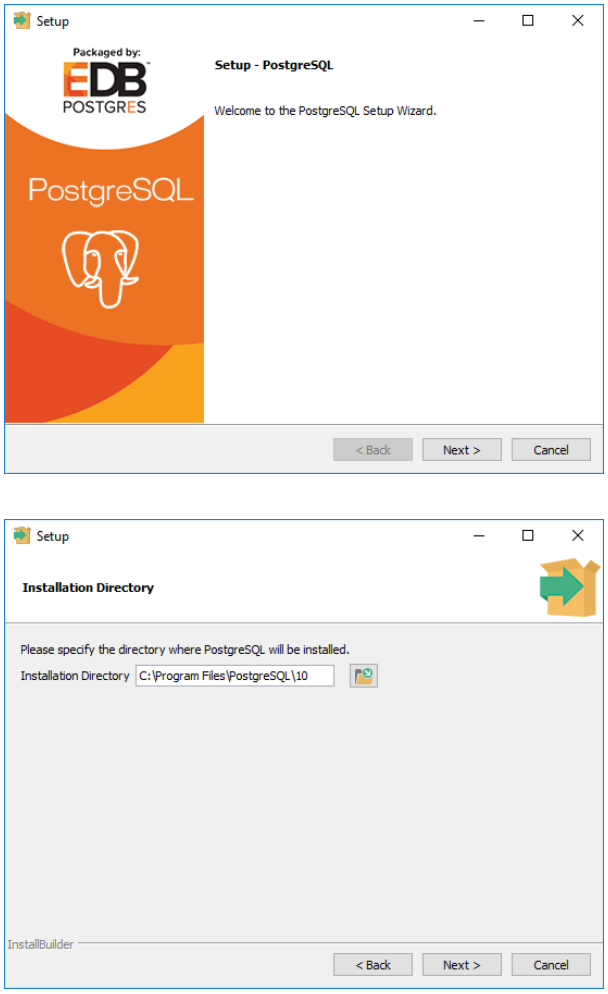
8.2 MS Windows

Ukážeme si doporučený postup instalace pomocí PostgreSQL from EnterpriseDB instalátoru.



Obr. 8.1: Zvolíme verzi PostgreSQL k instalaci.

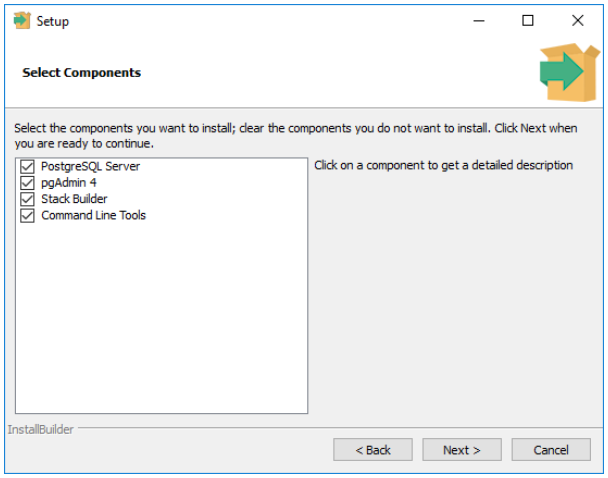
Poznámka: Instalátor může ještě před svým startem vynutit instalaci *Microsoft Visual C++ Redistributable*, pokud není na hostitelském počítači dostupný.



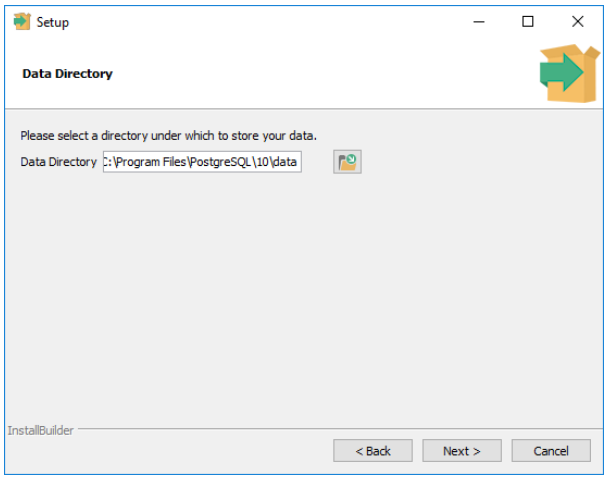
Obr. 8.2: Určíme adresář na disku, kam se PostgreSQL nainstaluje.

Poznámka: V PostGIS lze pracovat i s rastrovými daty, viz školení PostGIS pro pokročilé. Pokud plánujeme s takovými daty pracovat, tak musíme tuto funkcionalitu aktivovat již při instalaci PostGIS.

Po úspěšné instalaci PostgreSQL a PostGIS spustíme aplikaci *PgAdmin 4*, pomocí které lze naimportovat školící databázi GISMentors.



Obr. 8.3: Doporučujeme nainstalovat všechny kompomenty včetně konzolových nástrojů a PgAdmin 4. Stack Builder je nutností pro navazující instalaci PostGIS.



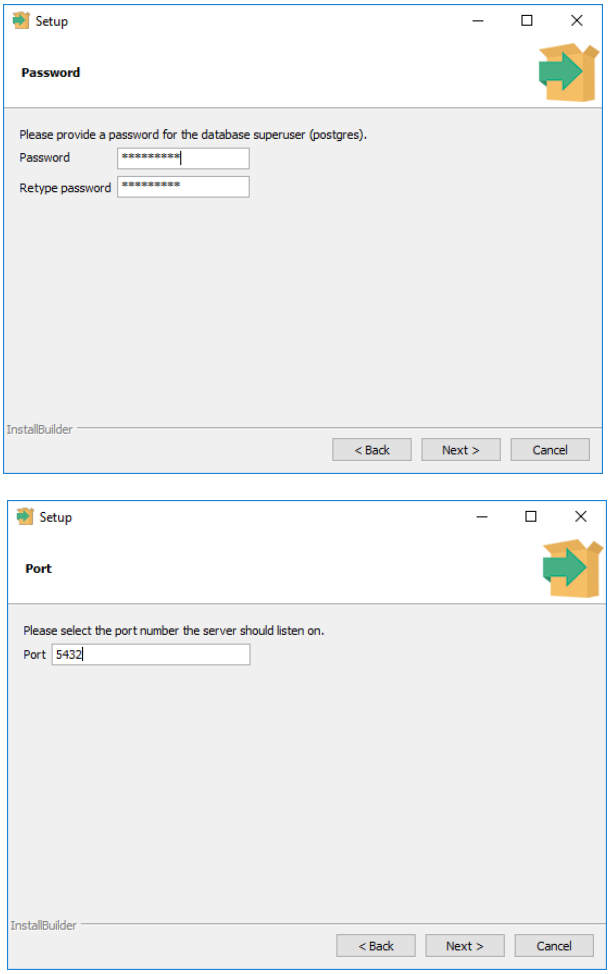
Obr. 8.4: V dalším kroku zvolíme adresář, kam se budou ukládat uživatelská data (mohou být velká podle toho k čemu budete databázi využívat).

8.3 Import databáze GISMentors

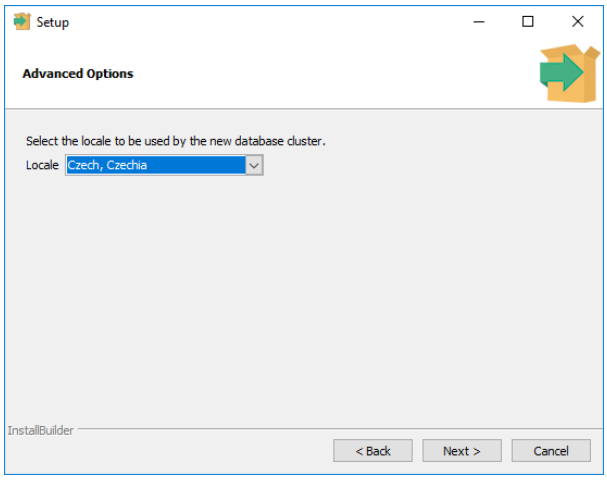
Soubor ke stažení: http://training.gismentors.eu/geodata/postgis/gismentors.dump

Databázi GISMentors lze naimportovat z grafické aplikace PgAdmin 4 anebo z příkazové řádky.

8.3.1 PgAdmin



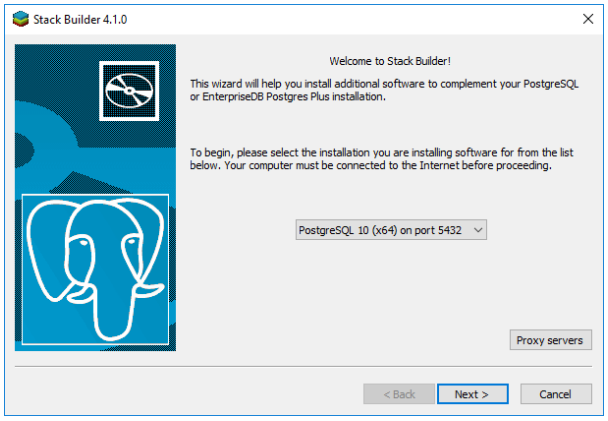
Obr. 8.5: Nastavíme heslo administrátora a port, na kterém databázový server poběží.



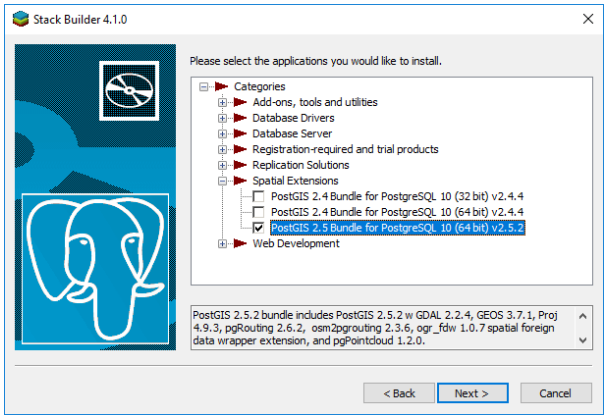
Obr. 8.6: Doporučujeme nastavit locales. Instalaci dokončíme, dalším krokem bude instalace PostGIS.

Důležité: Tato volba je důležitá, jinak skončí import chybou!

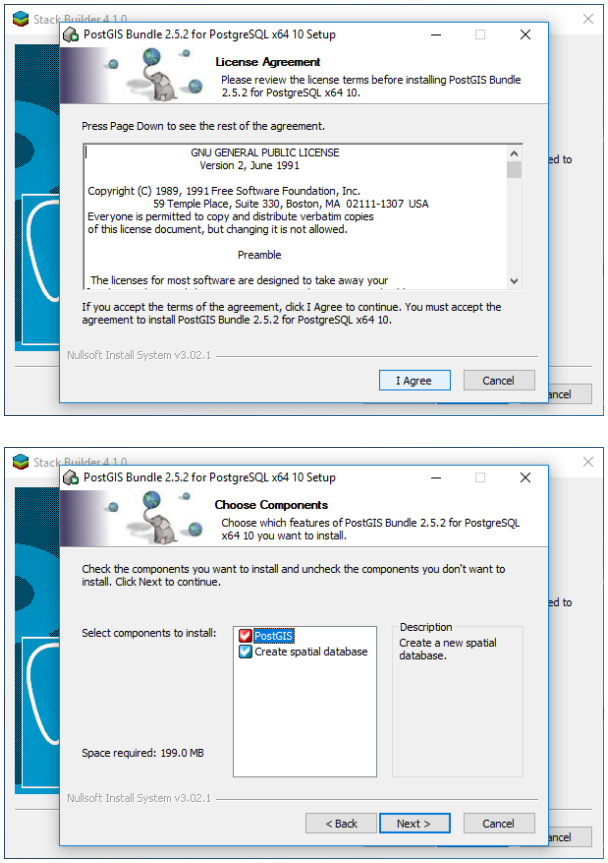
Obr. 8.7: Nástroj *StackBuilder* se spustí automaticky.



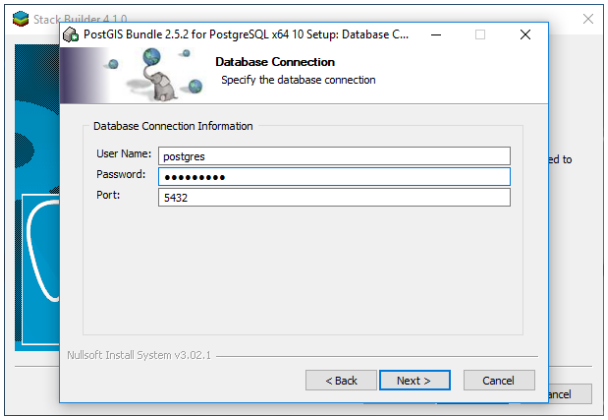
Obr. 8.8: Zvolíme databázový server, do kterého chceme doinstalovat PostGIS.



Obr. 8.9: V prostředí Stack Builderu v sekci Spatial Extensions zvolíme verzi PostGIS podle toho, zda jste nainstalovali 32 anebo 64bitovou verzi PostgreSQL. Adresář s nainstalovaným PostgreSQL by měl instalátor detekován automaticky.



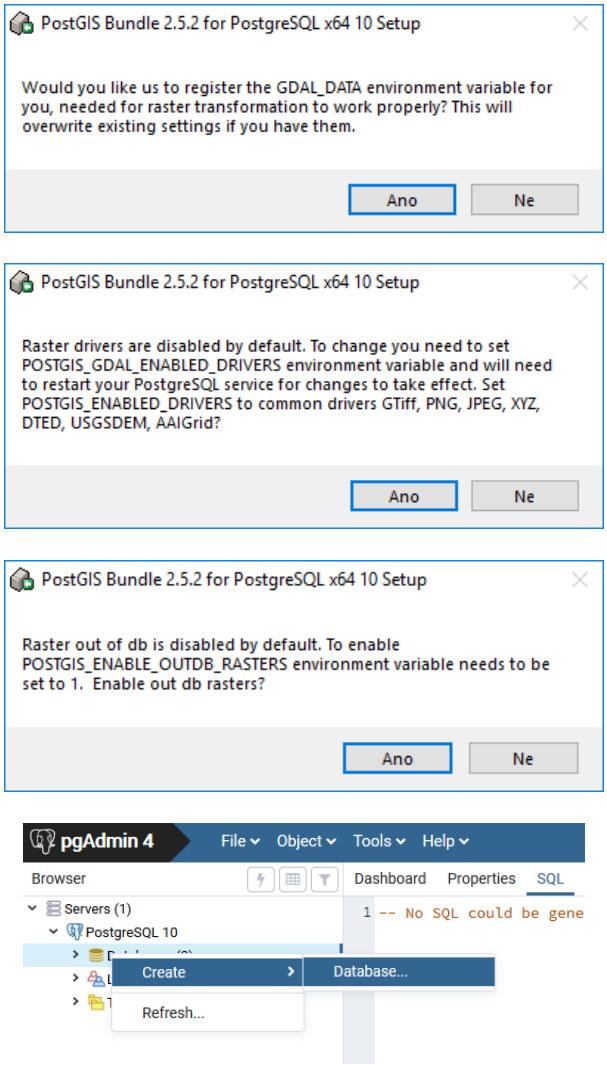
Obr. 8.10: Volitelně můžete vytvořit databázi PostGIS se vzorovými daty. Tento krok není ale nutný.



Obr. 8.11: Před samotnou instalací PostGIS musíme zadat již dříve definováné administrátorské heslo, viz Obr. 8.2.

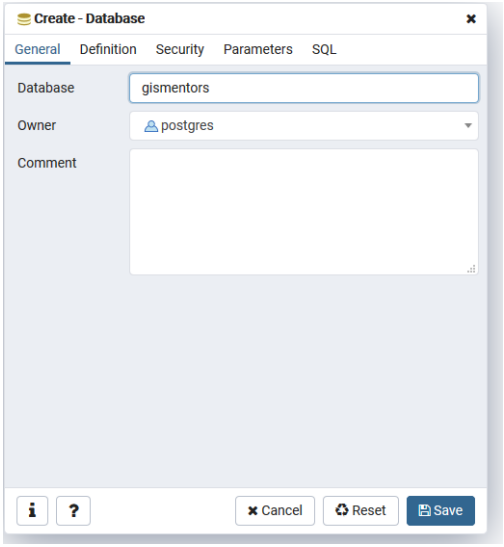
8.3.2 Z příkazové řádky

```
wget http://training.gismentors.eu/geodata/postgis/gismentors.dump
createdb gismentors
pg_restore gismentors.dump | psql gismentors
```

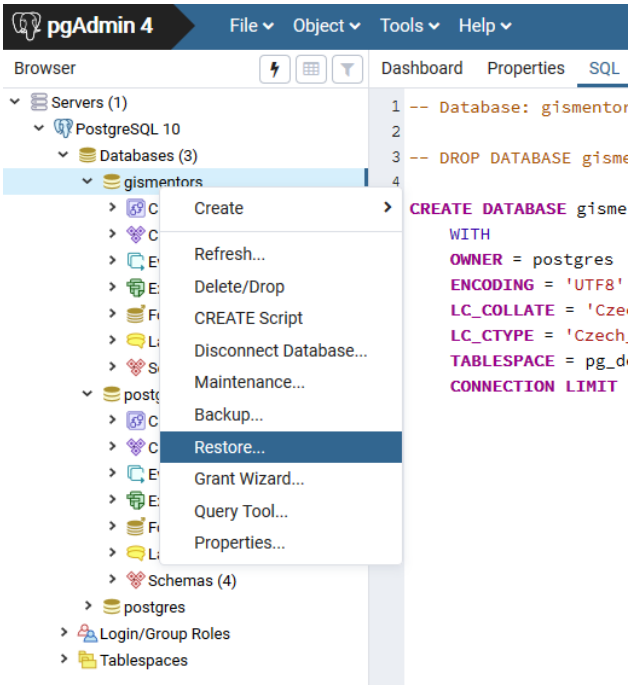


Obr. 8.12: Po připojení k databázovému serveru vytvoříme novou databázi.

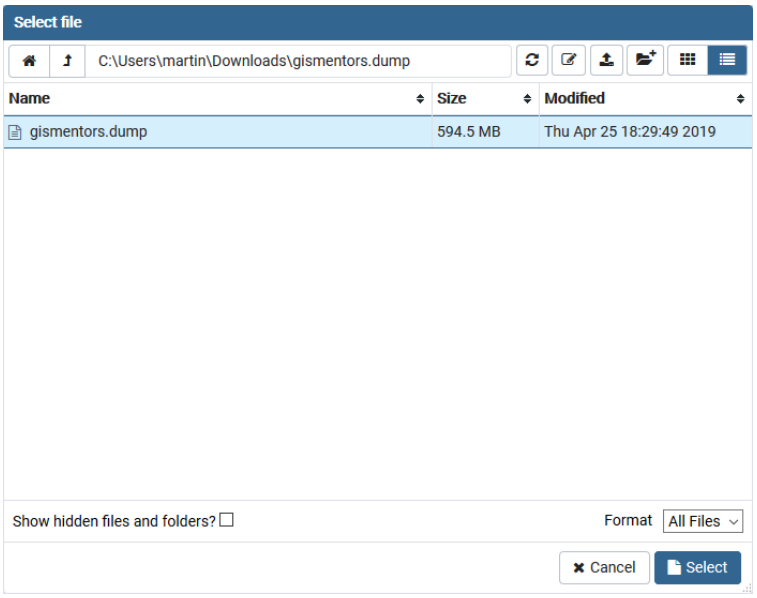
Tip: Kompletní skript pro Bash ke stažení zde.

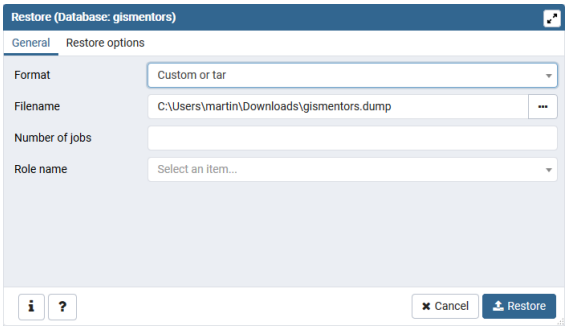


Obr. 8.13: Databázi nazveme „gismentors“.

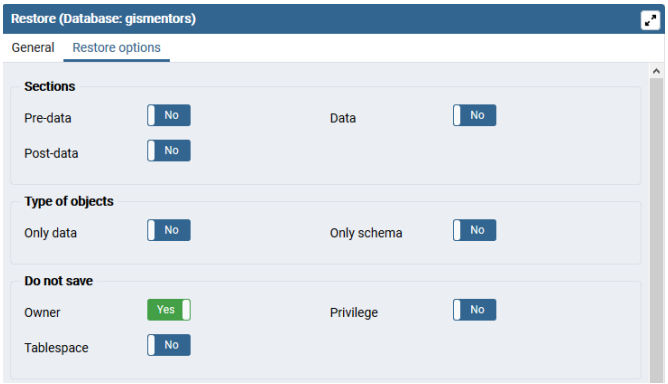


Obr. 8.14: Z kontextového menu nad databází zvolíme funkci „Obnovit“.

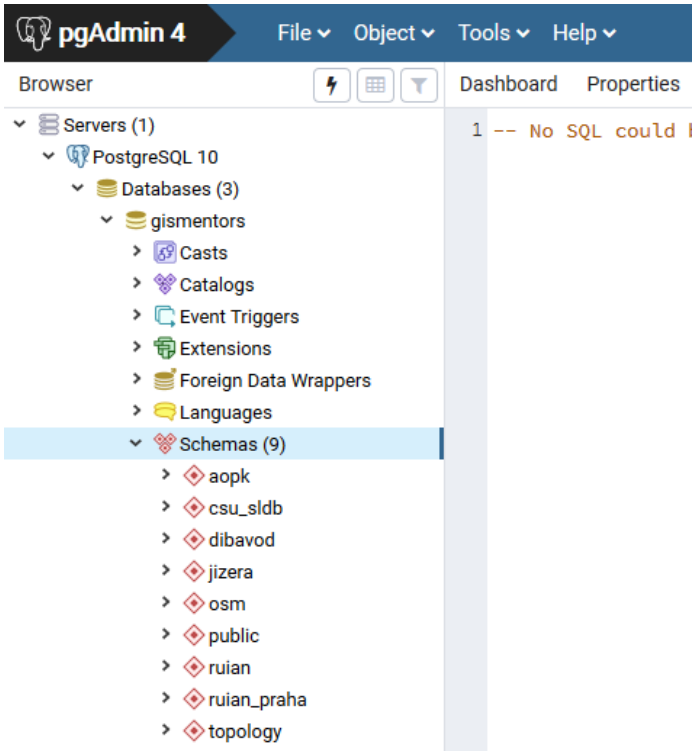




Obr. 8.15: V dialogu pro výběr souboru zvolíme dávku „gismentors.dump“ (je potřeba nastavit masku na „All files“).



Obr. 8.16: V sekci „Volby obnovení“ zaškrtěte „Do not save - owner“.


pgAdmin 4
File
Object
Tools
Help
Browser
Dashboard
Properties
Servers (1)
PostgreSQL 10
Databases (3)
gismentors
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Schemas (9)
aopk
csu_sldb
dibavod
jizera
osm
public
ruian
ruian_praha
topology
1 -- No SQL could

KAPITOLA 9

Dodatky

9.1 O dokumentu

Verze textu dokumentu: 0.6beta (sestaveno 25.09.2019)

9.1.1 Autoři

Za GISMentors:

- Jan Michálek <godzilalalala gmail.com>
- Martin Landa <martin.landa opengeolabs.cz>
- Jan Růžička <jan.ruzicka opengeolabs.cz>

9.1.2 Text dokumentu

Online HTML verze textu školení je dostupná na adrese:

- http://training.gismentors.eu/postgis-zacatecnik

Zdrojové texty školení jsou dostupné na adrese:

- https://github.com/GISMentors/postgis-zacatecnik

Rejstřík

S

T

U

V

Z