МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)
Кафедра программной инженерии

Пространственные базы данных

Методические указания к практическим занятиям
по дисциплине «Пространственные базы данных»
для студентов направления подготовки
09.04.04 «Программная инженерия», реализующегося по модели элитного обучения

Курск 2024

УДК 004.65

Составитель Е.И.Аникина

Рецензент
Кандидат технических наук, доцент кафедры программной инженерии *Е.А.Петрик*

Пространственные базы данных: методические указания к практическим занятиям по дисцитическимплине «Пространственные базы данных» для студентов направления подготовки 09.04.04 «Программная инженерия»/Юго-Зап. гос. ун-т; сост. Е.И. Аникина. Курск, 2024. 44 с.

Содержит задания к практическим работам, теоретические сведения и примеры решения задач по теме курса, связанной с разработкой интерфейса пользователя и реализацией функционала приложения для работы с пространсвенной базой данных.

Предназначено для студентов направления подготовки 09.04.04 «Программная инженерия».

Текст печатается в авторской редакции.

Подписано в печать . Формат 60x84 1/16.
Усл. печ. л. . Уч.-изд. л. . Тираж 100 экз. Заказ .
Бесплатно.

Юго-Западный государственный университет
305040, Курск, ул.50 лет Октября, 94.

Практическое занятие №1

Проектирование и создание пространственной базе данных

Практическое занятие №2

Проектирование и реализация пространственных запросов

Практическое занятие №3

Анализ и визуализация пространственных данных

Практическое занятие №1

ПРОЕКТИРОВАНИЕ И СОЗДАНИЕ ПРОСТРАНСТВЕННОЙ БАЗЫ ДАННЫХ

Задание

1. Изучите основные понятия реляционной модели данных: отношение, атрибуты (поля) отношения, кортежи отношения, индивидуальные идентификаторы.
2. Изучите основы теории нормальных форм отношений в реляционной модели данных. Запишите в отчет определения первой, второй и третьей нормальных форм.
3. Проанализируйте объекты построенной вами ER-модели данных и определите, В какой из нормальных форм находится каждый из объектов?
Приведите доказательство в подтверждение ваших выводов.
4. Если на данном этапе проектирования в вашей ER-модели остались обьекты, не находящиеся в 3НФ, проведите процедуру нормализации. Все обьекты вашей ER-модели надо привести к третьей нормальной форме.
Опишите процедуру нормализации.
6. Если в вашей ER-модели произошли изменения в результате нормализации, то внесите изменения в ER-диаграмму с помощью программы Oracle SQL Data Modeler

Пример

Проанализируем объекты построенной ER-модели данных и определим, в какой из нормальных форм находится каждый из объектов.
ER-модель данных поазана на рисунке 1.

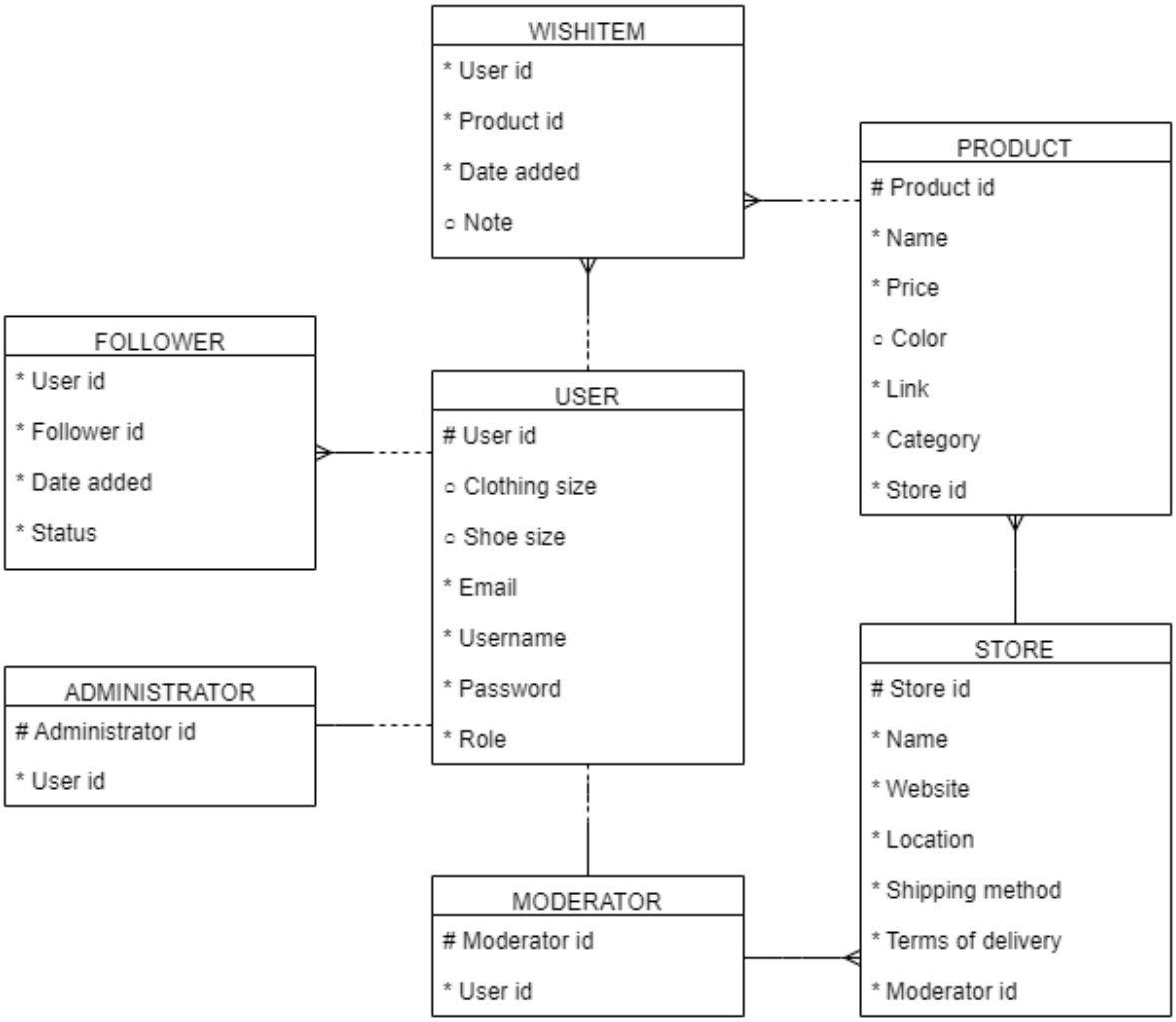


Рисунок 1 – ER-модель данных

Все объекты построенной ER-модели данных должны находиться в третьей нормальной форме.

Первая нормальная форма (1NF)

Все атрибуты объекта ПОЛЬЗОВАТЕЛЬ (USER) имеют только одно значение.

Все атрибуты объекта МАГАЗИН (STORE) имеют только одно значение.

Все атрибуты объекта ТОВАР (PRODUCT) имеют только одно значение.

Все атрибуты объекта ЭЛЕМЕНТ СПИСКА ЖЕЛАНИЙ (WISHITEM) имеют только одно значение.

Все атрибуты объекта ЭЛЕМЕНТ СПИСКА ОТСЛЕЖИВАНИЯ (FOLLOWER) имеют только одно значение.

Все атрибуты объекта АДМИНИСТРАТОР (ADMINISTRATOR) имеют только одно значение.

Все атрибуты объекта МОДЕРАТОР (MODERATOR) имеют только одно значение.

Вторая нормальная форма (2NF)

Атрибуты объекта МАГАЗИН (STORE) зависят от полного UID (id магазина).

Атрибуты объекта ТОВАР (PRODUCT) зависят от полного UID (id товара).

Атрибуты объекта ПОЛЬЗОВАТЕЛЬ (USER) зависят от полного UID (id пользователя).

Атрибуты объекта АДМИНИСТРАТОР (ADMINISTRATOR) зависят от полного UID (id администратора).

Атрибуты объекта МОДЕРАТОР (MODERATOR) зависят от полного UID (id модератора).

Третья нормальная форма (3NF)

При проверке всех объектов не было выявлено независимых от UID атрибутов или атрибутов, зависимых от других атрибутов.

На основе ER-модели данных в программе dbdiagram построена реляционная модель данных, показанная на рисунке 2.

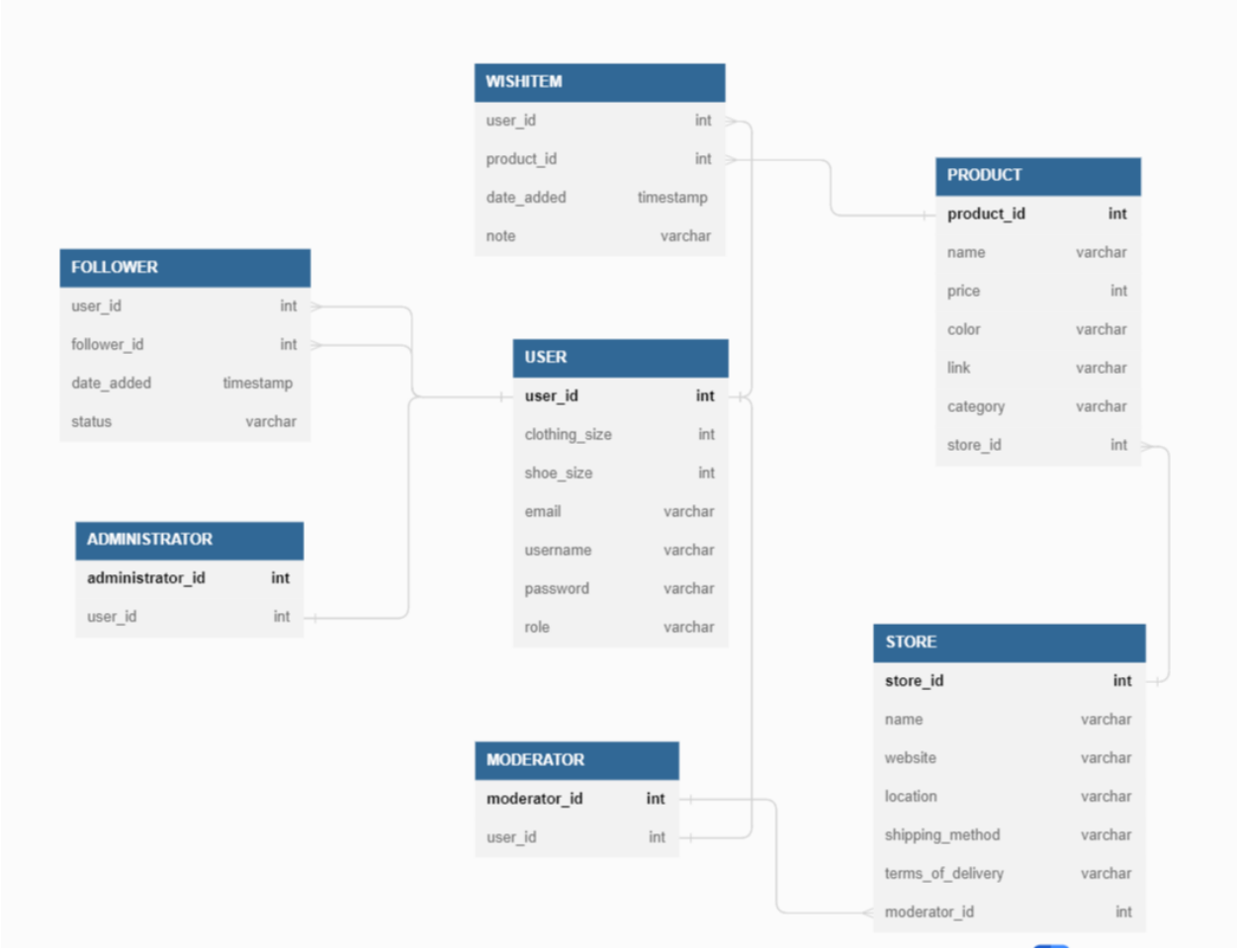


Рисунок 2 – Реляционная модель данных

Названия таблиц и столбцов каждой таблицы приведены в соответствии с требованиями к именам MySQL.

В таблице 2 представлена структура таблицы пользователей.

Таблица 2 – Пользователи

Name			
USER			
Field	**Type**	**Null**	**Key**
user_id	int	NO	PRI
clothing_size	int	NO	NULL
shoe_size	int	YES	NULL
email	varchar(255)	NO	UNI
username	varchar(255)	NO	UNI
password	varchar(255)	NO	NULL
role	varchar(255)	NO	NULL

В таблице 3 представлена структура таблицы магазинов.

Таблица 3 – Магазины

Name			
STORE			
Field	**Type**	**Null**	**Key**
store_id	int	NO	PRI
name	varchar(255)	NO	UNI
website	varchar(255)	NO	UNI
location	varchar(255)	NO	NULL
shipping_method	varchar(255)	NO	NULL
terms_of_delivery	varchar(255)	NO	NULL
moderator_id	int	NO	MUL

В таблице 4 представлена структура таблицы товаров.

Таблица 4 – Товары

Name			
PRODUCT			
Field	**Type**	**Null**	**Key**
product_id	int	NO	PRI
name	varchar(255)	NO	NULL
price	int	NO	NULL
color	varchar(255)	YES	NULL
link	varchar(255)	NO	NULL
category	varchar(255)	NO	NULL
store_id	int	NO	MUL

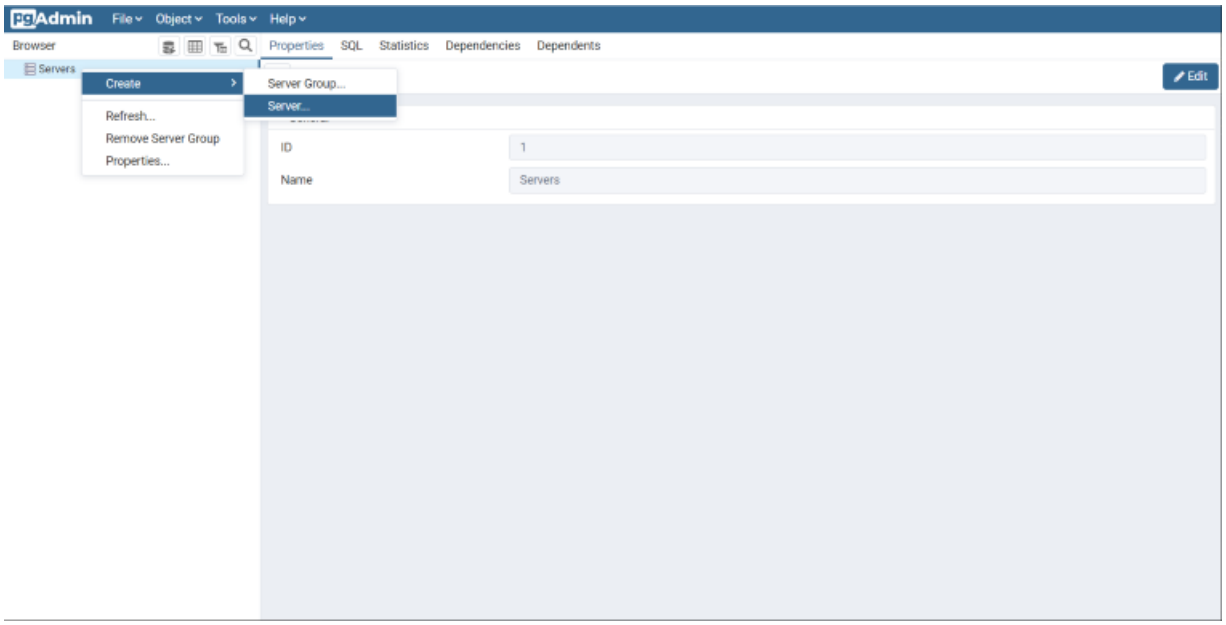
В таблице 5 представлена структура таблицы элементов списка желаний.

Создание пространственной базы данных средствами pgAdmin

https://postgis.net/workshops/postgis-intro/index.html

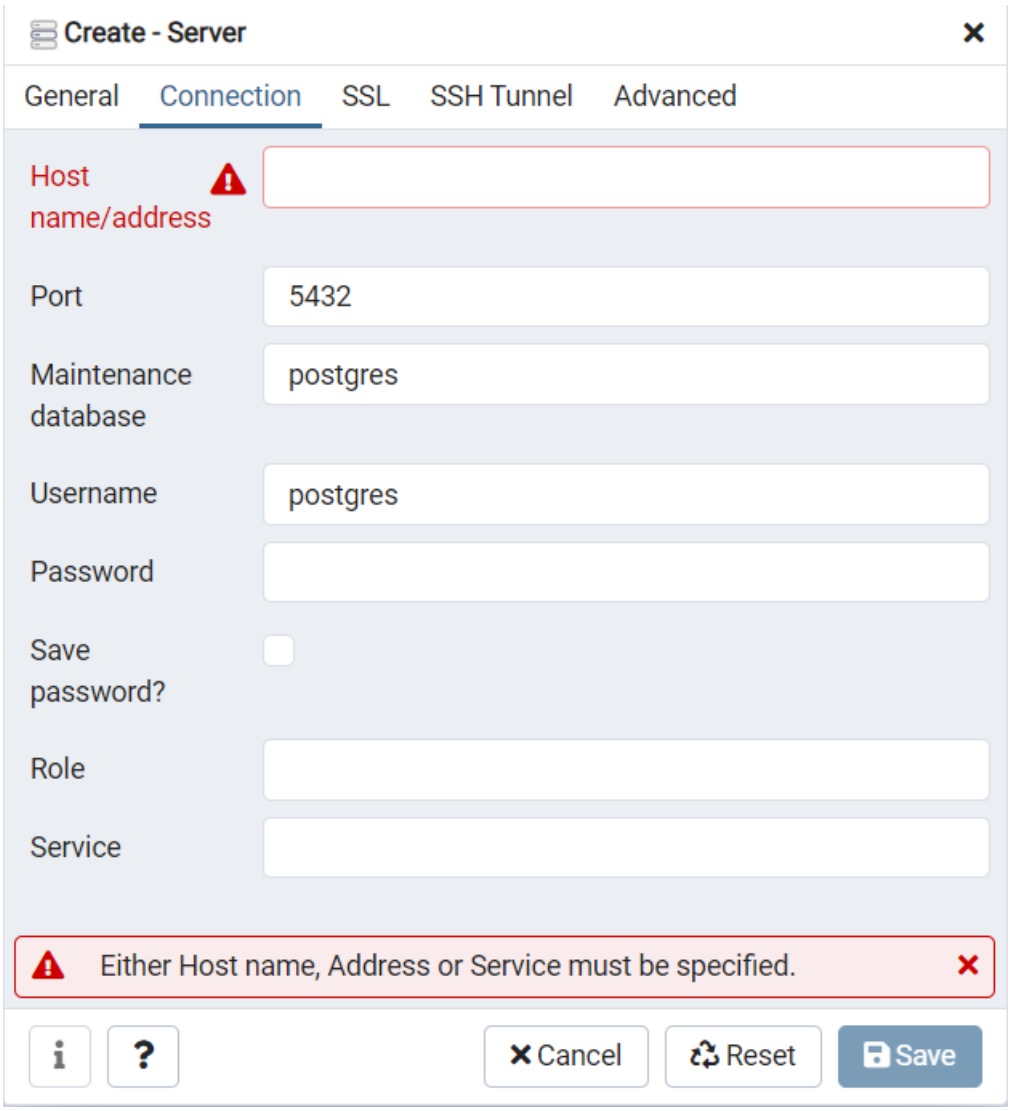
PgAdmin - это популярная графическая среда для работы с PostgreSQL, которая включает в себя средство просмотра геометрии (a geometry viewer), которое можно использовать для просмотра пространственных запросов в PostGIS.

1. Найдите pgAdmin и запустите его.



2. Если вы запускаете pgAdmin впервые, возможно, у вас еще не настроены серверы. Щелкните правой кнопкой мыши элемент Серверы (`Servers`).

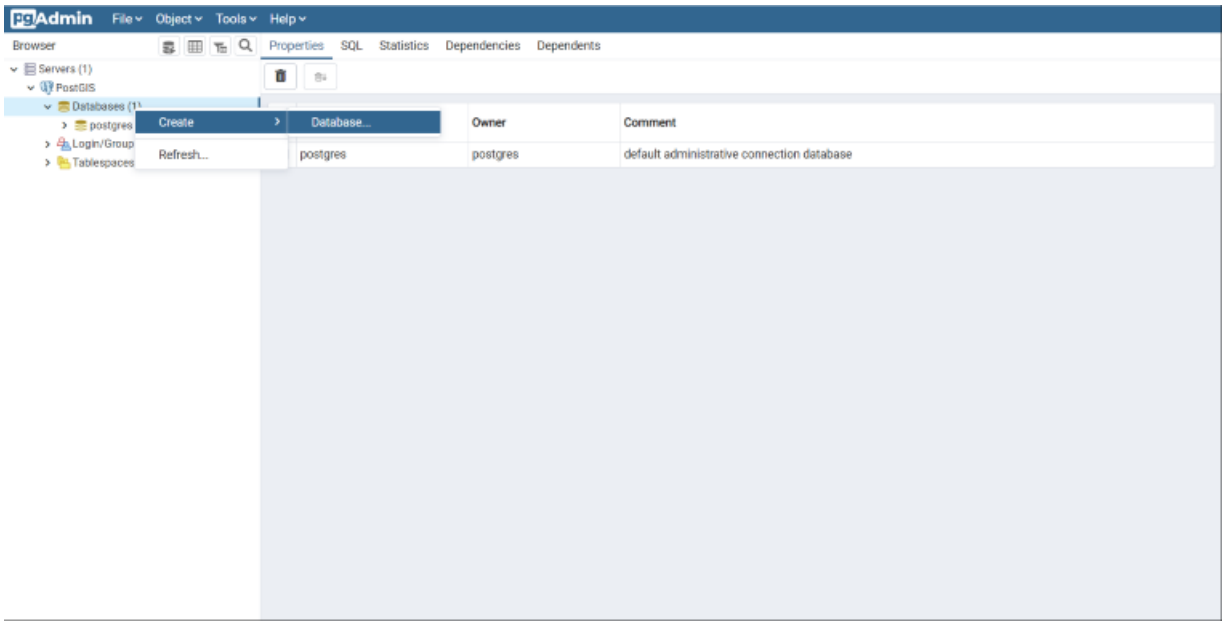
Назовем наш сервер PostGIS. На вкладке «Соединение» (Connection) введите имя/адрес хоста (`Host name/address`). Если вы работаете с локальной установкой PostgreSQL, вы сможете использовать localhost. Если вы используете облачный сервис, вы сможете получить имя хоста из своей учетной записи.
Оставьте для порта значение 5432, а для базы данных обслуживания (Maintenance) и имени пользователя — postgres. Пароль должен быть тем, который вы указали при локальной установке или в облачном сервисе.



4.2. Создание базы данных

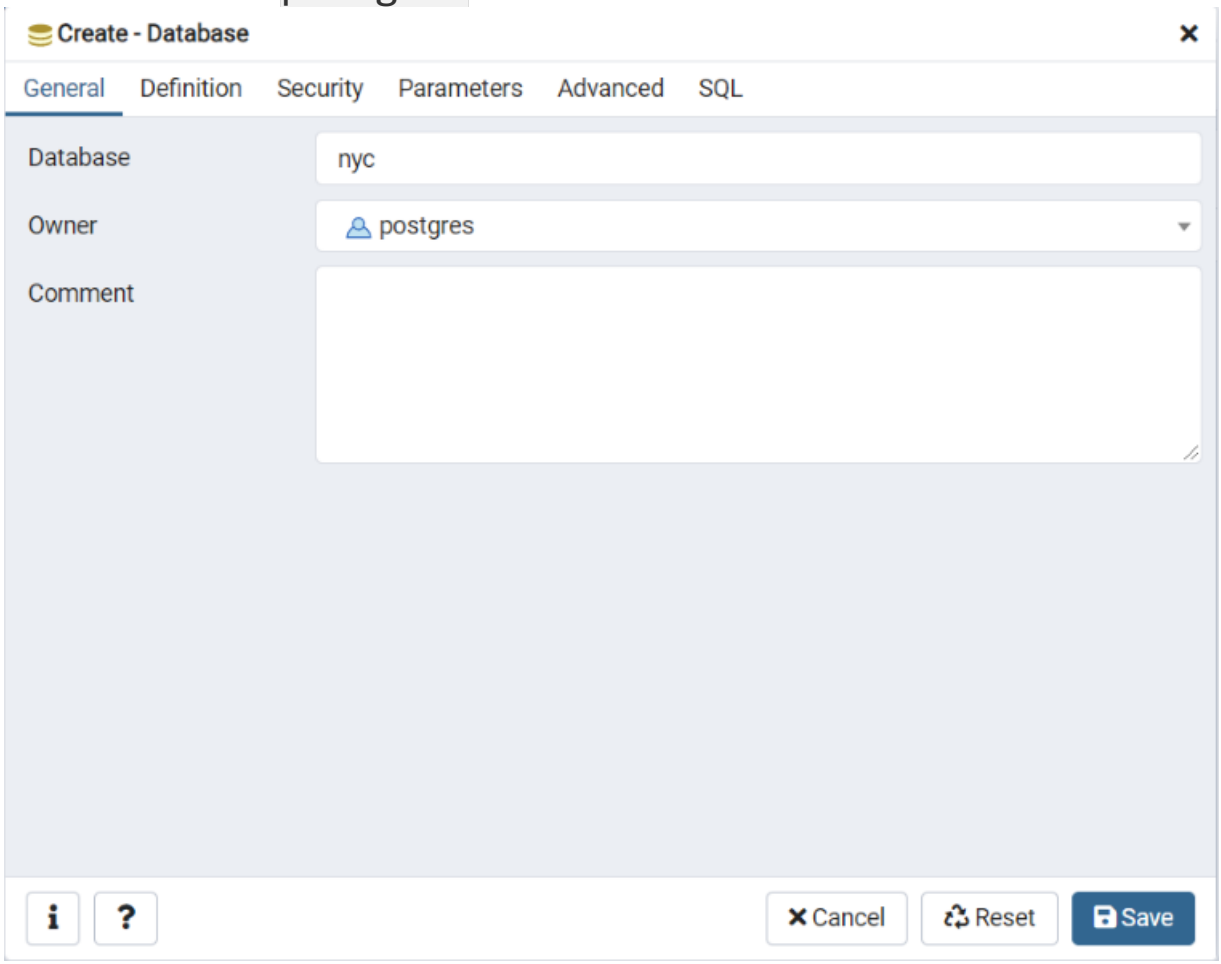
1. 1. Откройте элемент дерева баз данных и посмотрите на доступные базы данных. База данных Postgres является пользовательской базой данных для пользователя Postgres по умолчанию и не слишком интересна для нас.

   2. Щелкните правой кнопкой мыши по элементу баз данных и выберите `New Database`.

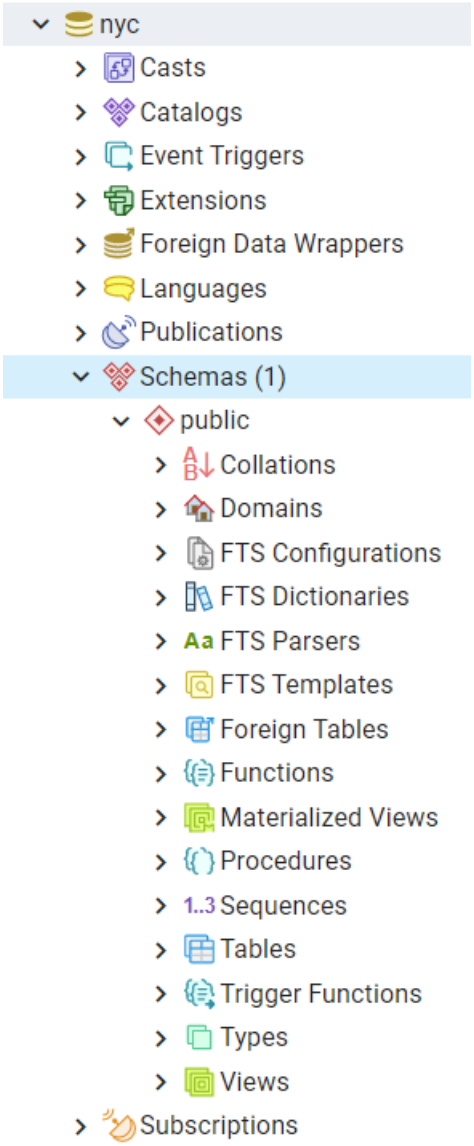


2. Заполните форму Create Database, как показано ниже, и нажмите OK.

Name `nyc`
Owner `postgres`



3. Выберите новую базу данных `nyc` и откройте ее, чтобы отобразить дерево объектов. Вы увидите схему `public`.



4. Нажмите кнопку SQL -запроса, показанную ниже (или перейдите к инструментам запроса Tools > Query Tool).

5. Введите следующий запрос в текстовое поле запроса, чтобы загрузить пространственное расширение PostGIS:

```
CREATE EXTENSION postgis;
```

6. Нажмите кнопку ***Play*** на панели инструментов (или нажмите F5), чтобы выполнить запрос.
7. Теперь подтвердите, что PostGIS установлен с помощью функции PostGIS:

```
SELECT postgis_full_version();
```

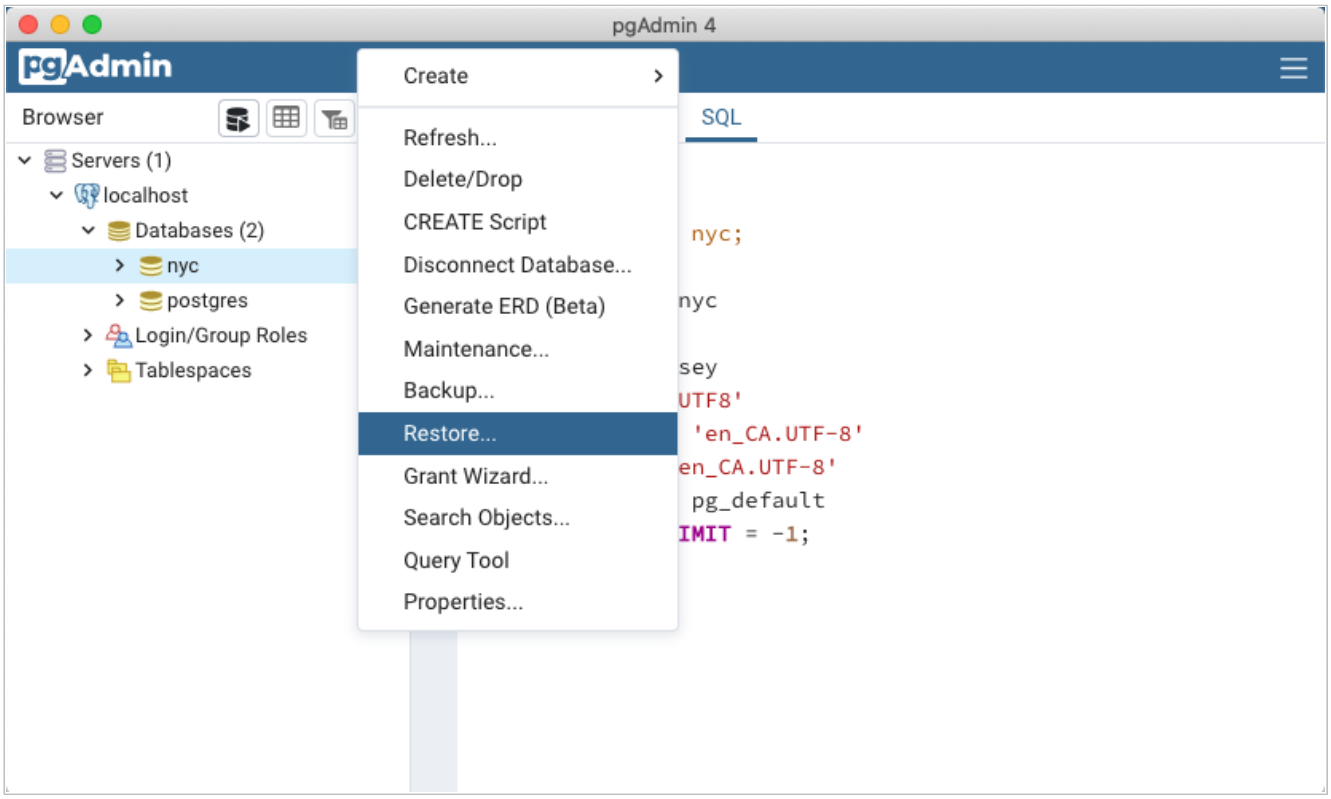
Вы успешно создали пространственную базу данных PostGIS !!

Загрузка пространственных данных

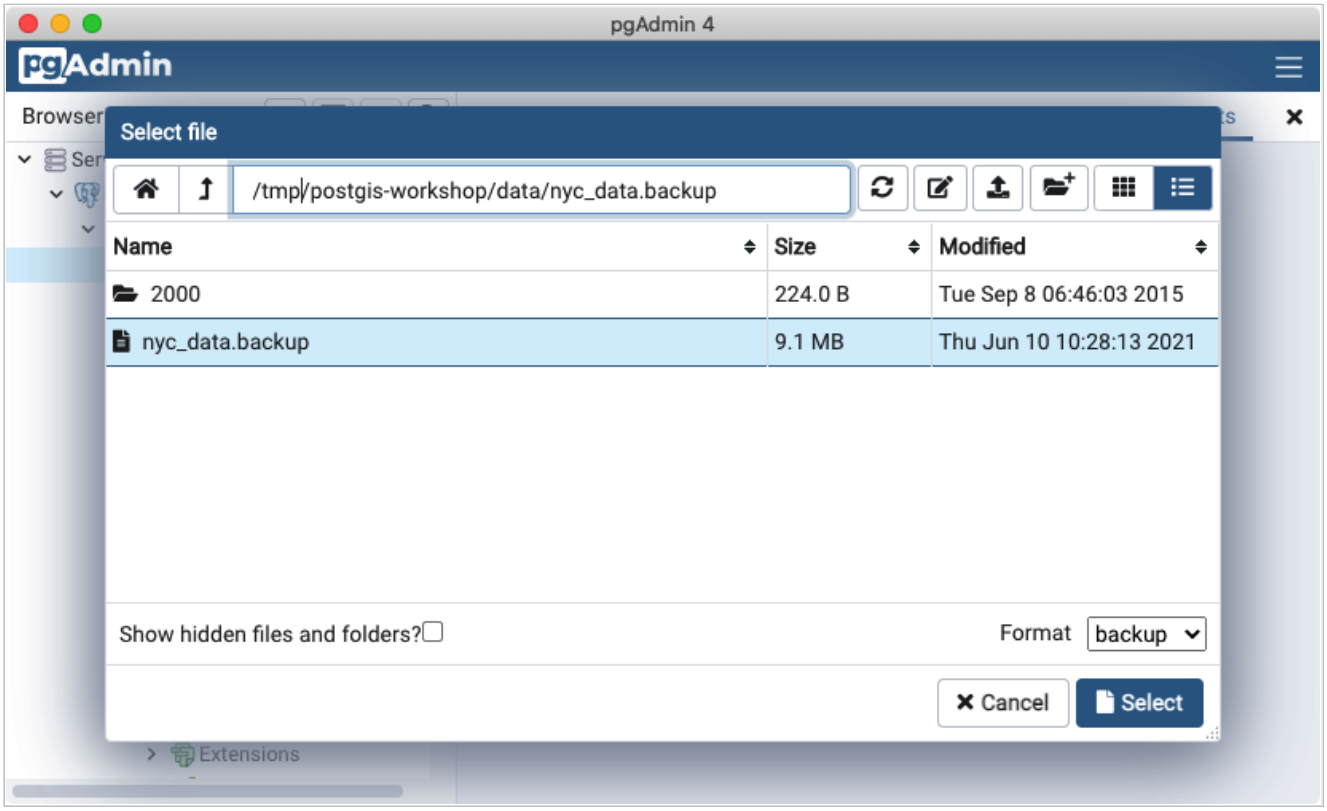
PostGIS поддерживается широким спектром библиотек и приложений и предоставляет много вариантов загрузки данных.

Мы загрузим данные для выполнения лабораторных работ из файла резервной копии базы данных (**Backup File) -** `nyc_data.backup.`

В браузере PgAdmin щелкните правой кнопкой мыши по значку базы данных `nyc`, а затем выберите опцию **Restore** ….

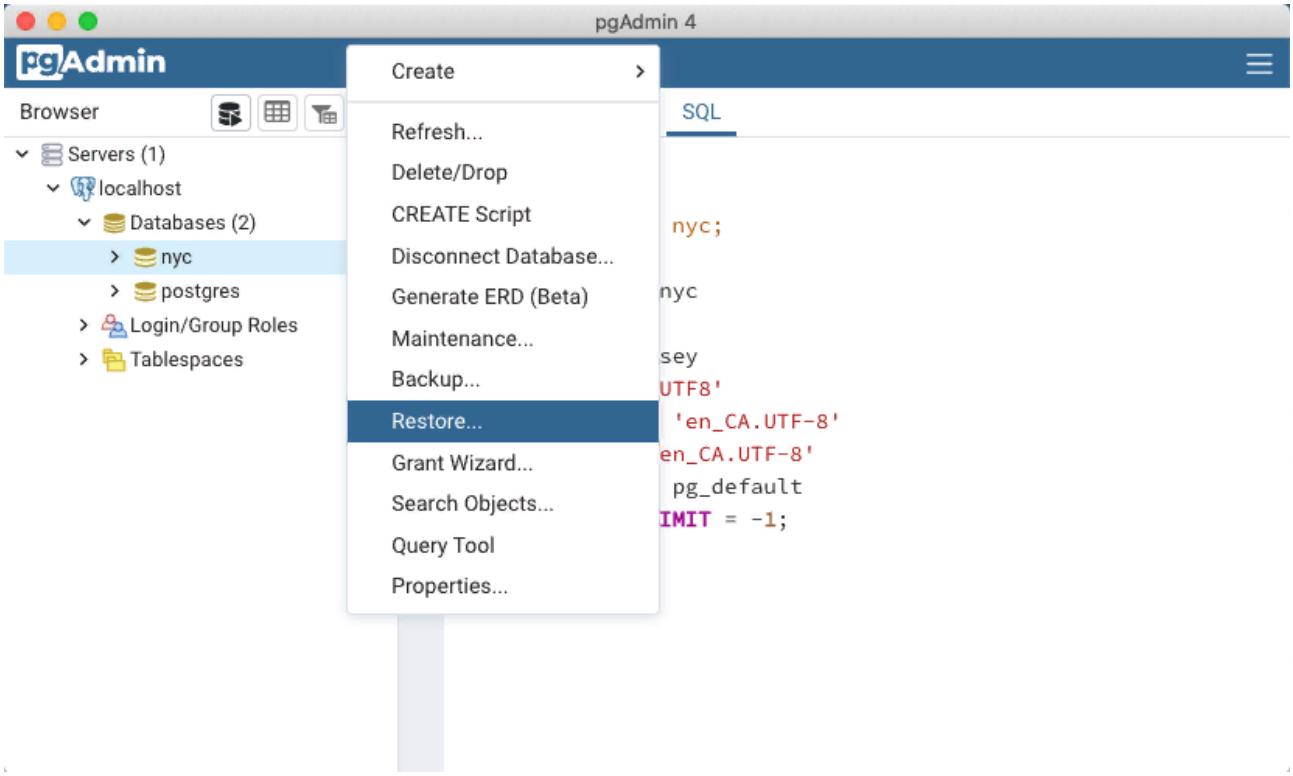


- Выберите ранее Скопированный файл nyc_data.backup.

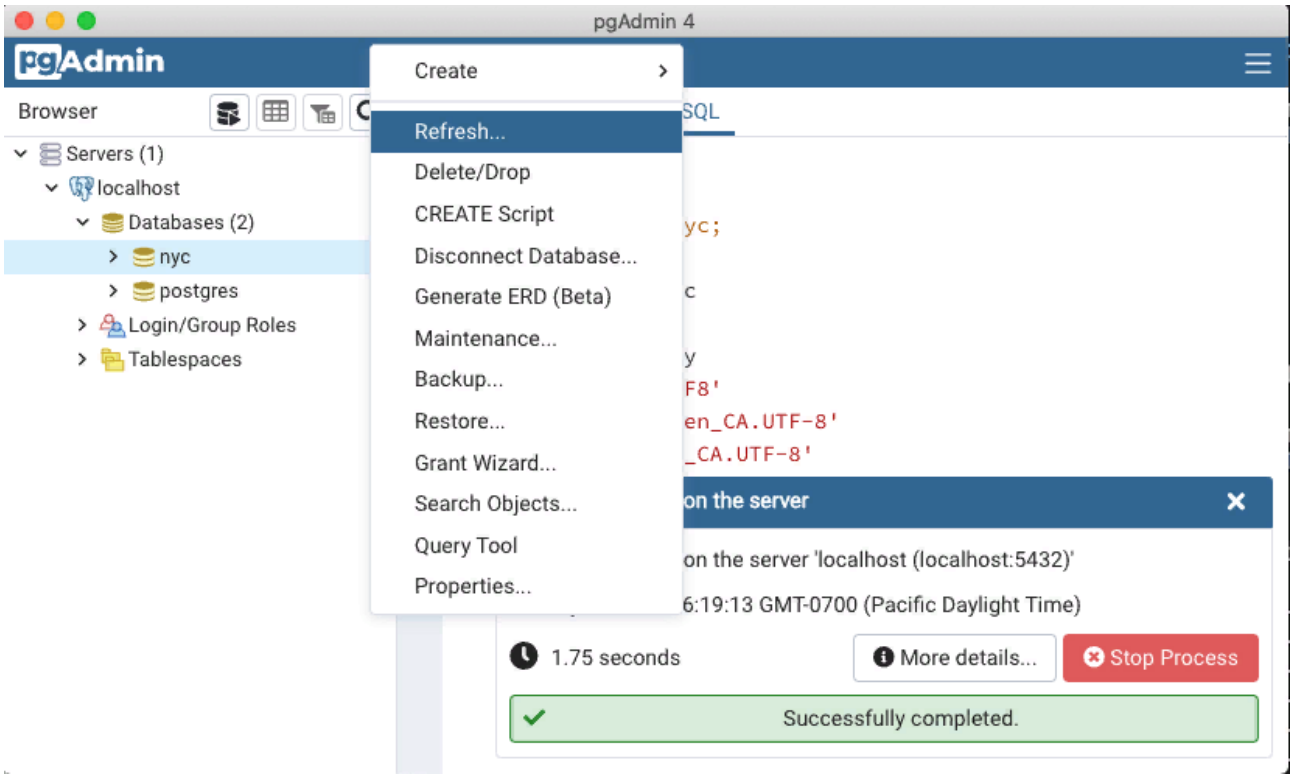


- Нажмите на вкладку «Параметры восстановления (Restore options)», прокрутите вниз до раздела «Не сохранить (Do not save)» и переключите владельца (Owner)на ***Да (***Yes).

- Нажмите на кнопку ***восстановить*** (Restore). Восстановление базы данных должно работать до завершения без ошибок.



- После завершения загрузки щелкните правой кнопкой мыши базу данных ***nyc*** и выберите опцию обновления (Refresh), чтобы обновить клиентскую информацию о том, какие таблицы существуют в базе данных.


pgAdmin 4
pgAdmin
Browser
Servers (1)
localhost
Databases (2)
nyc
postgres
Login/Group Roles
Tablespaces
Create
Refresh...
Delete/Drop
CREATE Script
Disconnect Database...
Generate ERD (Beta)
Maintenance...
Backup...
Restore...
Grant Wizard...
Search Objects...
Query Tool
Properties...
SQL
yc;
c
y
F8'
en_CA.UTF-8'
_CA.UTF-8'
on the server
on the server 'localhost (localhost:5432)'
6:19:13 GMT-0700 (Pacific Daylight Time)
1.75 seconds
More details...
Stop Process
Successfully completed.

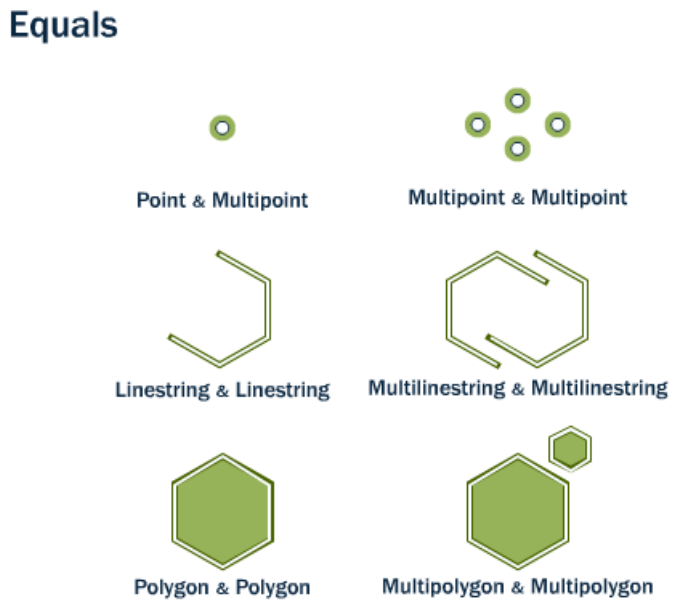
ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ ПРОСТРАНСТВЕННЫХ ЗАПРОСОВ

Пространственные отношения

До сих пор мы использовали только пространственные функции, которые измеряют (ST_Area, ST_Length), сериализуют (ST_GeomFromText) или десериализуют (ST_AsGML) геометрию. Общим для этих функций является то, что они одновременно работают только с одной геометрией.
Пространственные базы данных являются мощным инструментом, поскольку они не только хранят геометрию, но и позволяют сравнивать отношения между геометриями.
Такие вопросы, как «Какие велосипедные стоянки находятся ближе всего к парку?» или «Где пересекаются линии метро и улицы?» На этот вопрос можно ответить только путем сравнения геометрии велосипедных стоек, улиц и линий метро.
Стандарт OGC определяет следующий набор методов сравнения геометрии.

ST_Equals

ST_Equals(geometry A, geometry B) проверяет пространственное равенство двух геометрических объектов.



ST_Equals возвращает TRUE, если два геометрических объекта одного типа имеют одинаковые значения координат x,y, т. е. если вторая форма равна (идентична) первой форме.
Сначала давайте получим представление точки из нашей таблицы nyc_subway_stations. Мы возьмем только запись ‘Broad St’.

```
SELECT name, geom
FROM nyc_subway_stations
WHERE name = 'Broad St';
   name    |                          geom
-----------+-----------------------------------------------------
 Broad St |
0101000020266900000EEBD4CF27CF2141BC17D69516315141
```

Затем подключите представление геометрии к тесту ST_Equals:

```
SELECT name
FROM nyc_subway_stations
WHERE ST_Equals(
  geom,

'0101000020266900000EEBD4CF27CF2141BC17D69516315141');
Broad St
```

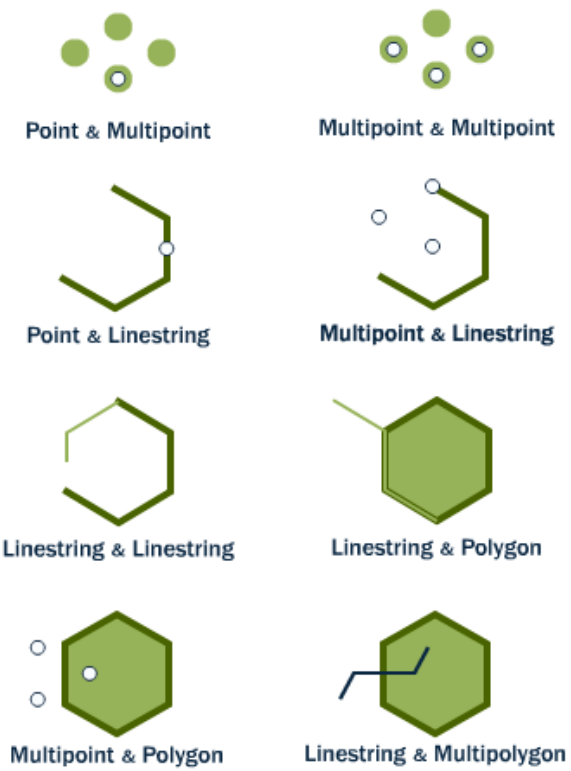
Note

Представление точки было не очень удобочитаемым. (`0101000020266900000EEBD4CF27CF2141BC17D69516315141`) но это было точное представление значений координат. Для проверки равенства необходимо использовать точные координаты.

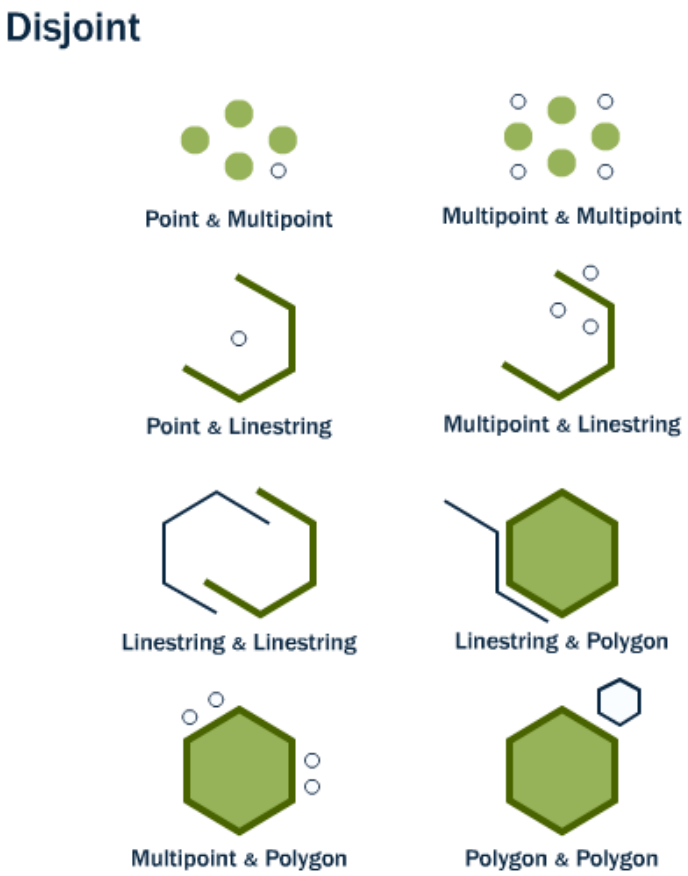
ST_Intersects, ST_Disjoint, ST_Crosses and ST_Overlaps

ST_Intersects, ST_Crosses, and ST_Overlaps проверить, пересекаются ли внутренние области геометрических фигур.

Intersects



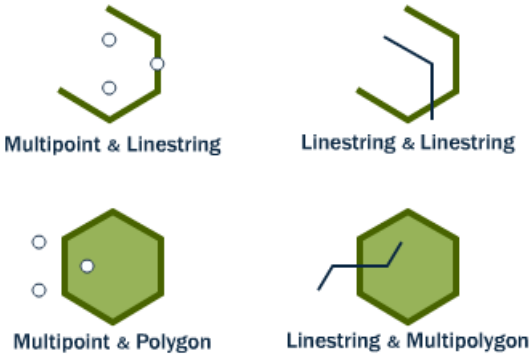
ST_Intersects(geometry A, geometry B) возвращает t (ИСТИНА), если две фигуры имеют какое-либо общее пространство, т. е. если их границы или внутренние области пересекаются.



The opposite of ST_Intersects is ST_Disjoint(geometry A , geometry B).

Зачастую более эффективно проверять «не пересекается», чем проверять «непересекающиеся», потому что тесты на пересечения могут быть пространственно индексированы, а тест на непересекающиеся — нет.

Cross



For multipoint/polygon, multipoint/linestring, linestring/linestring, linestring/polygon, and linestring/multipolygon comparisons, ST_Crosses(geometry A, geometry B) returns t (TRUE) если в результате пересечения образуется геометрия, размерность которой на единицу меньше максимальной размерности двух исходных геометрий, и набор пересечений является внутренним для обеих исходных геометрий.

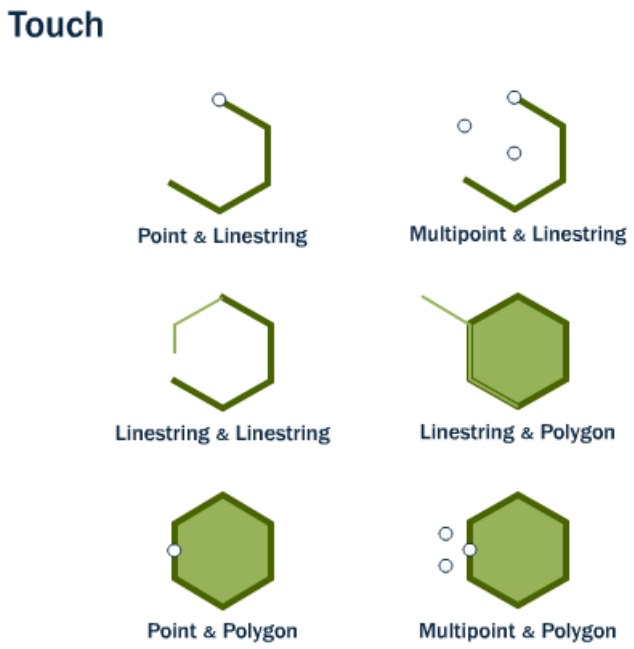
Overlap

ST_Overlaps(geometry A, geometry B) сравнивает две геометрии одного и того же измерения и возвращает TRUE, если их набор пересечений приводит к получению геометрии, отличной от обеих, но одного и того же измерения.
Давайте возьмем нашу станцию метро Broad Street и определим ее район с помощью функции ST_Intersects:

```
SELECT name, ST_AsText(geom)
FROM nyc_subway_stations
WHERE name = 'Broad St';
POINT(583571 4506714)
SELECT name, boroname
FROM nyc_neighborhoods
WHERE ST_Intersects(geom,
ST_GeomFromText('POINT(583571 4506714)',26918));
         name         | boroname
----------------------+----------
-
 Financial District | Manhattan
```

ST_Touches

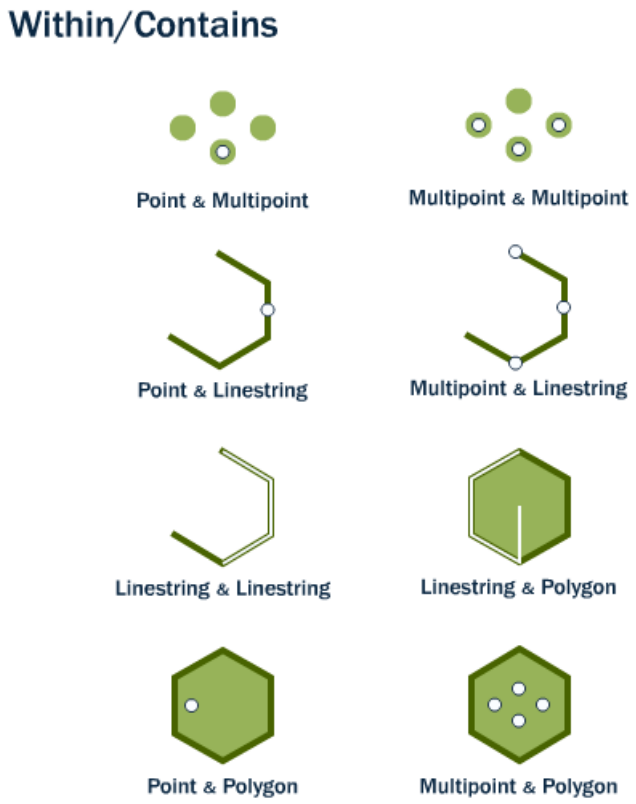
ST_Touches проверяет, касаются ли две геометрии своими границами, но не пересекаются внутри



ST_Touches(geometry A, geometry B) возвращает TRUE, если любая из границ геометрии пересекается или если только одна из внутренних частей геометрии пересекает границу другой.

ST_Within and ST_Contains

ST_Within and ST_Contains проверит, полностью ли одна геометрия находится внутри другой.



ST_Within(geometry A , geometry B) возвращает TRUE, если первая геометрия полностью находится внутри второй геометрии. ST_Within проверяет результат, прямо противоположный ST_Contains.
ST_Contains(геометрия A, геометрия B) возвращает TRUE, если вторая геометрия полностью содержится в первой геометрии.

ST_Distance and ST_DWithin

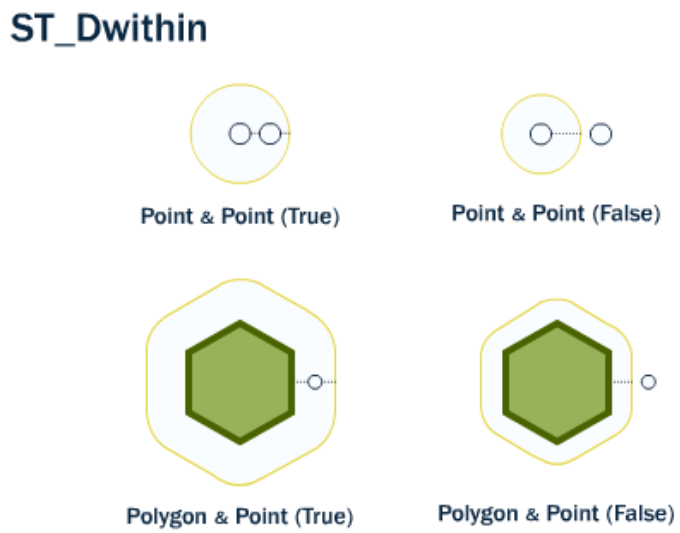
Чрезвычайно распространенный вопрос ГИС: «Найти все, что находится на расстоянии X от этого другого».
ST_Distance(геометрия A, геометрия B) вычисляет кратчайшее расстояние между двумя геометриями и возвращает его в виде числа с плавающей запятой. Это полезно для фактического сообщения о расстоянии между объектами.

```
SELECT ST_Distance(
```

```
  ST_GeometryFromText('POINT(0
5)'),

ST_GeometryFromText('LINESTRING
(-2 2, 2 2)'));
3
```

Для проверки того, находятся ли два объекта на расстоянии друг от друга, функция ST_DWithin обеспечивает ускоренный индексом тест истинного/ложного значения. Это полезно для таких вопросов, как «сколько деревьев находится в радиусе 500 метров от дороги?». Вам не нужно рассчитывать реальный буфер, вам просто нужно проверить соотношение расстояний.



Снова воспользовавшись станцией метро Broad Street, мы можем найти улицы рядом (в пределах 10 метров) со станцией метро:

```
SELECT name
FROM nyc_streets
```

```
WHERE ST_DWithin(
        geom,
        ST_GeomFromText('POINT(583571
4506714)',26918),
        10
      );
     name
--------------
   Wall St
   Broad St
   Nassau St
```

И мы можем проверить ответ на карте. Станция Broad St на самом деле находится на пересечении улиц Уолл, Брод и Нассау.

УПРАЖНЕНИЯ

3. Форму Какого типа геометричеой фигуры имеет улица под названием Atlantic Commons?

```
SELECT ST_AsText(geom)
  FROM nyc_streets
  WHERE name = 'Atlantic Commons';

MULTILINESTRING((586781.701577724
4504202.15314339,586863.51964484 4504215.9881701))
```

4. В каком районе и районе находится Atlantic Commons?

```
SELECT name, boroname
FROM nyc_neighborhoods
WHERE ST_Intersects(
  geom,
  ST_GeomFromText('LINESTRING(586782
4504202,586864 4504216)', 26918)
);
     name     | boroname
--------------+----------
 Fort Green | Brooklyn
```

5.

6. Note

«А почему сменили «MULTILINESTRING» на «LINESTRING»?» Пространственно они описывают одну и ту же форму, поэтому переход от одноэлементной мультигеометрии к одноэлементной позволяет сэкономить несколько нажатий клавиш.

Что еще более важно, мы также округлили координаты, чтобы их было легче читать, что на самом деле меняет результаты: мы не могли использовать предикат

ST_Touches(), чтобы узнать, какие дороги присоединяются к Atlantic Commons, потому что координаты уже не совсем те же самые.

• С какими улицами соединяется Atlantic Commons?

```
SELECT name
FROM nyc_streets
WHERE ST_DWithin(
  geom,
  ST_GeomFromText('LINESTRING(586782
4504202,586864 4504216)', 26918),
  0.1
);
      name
------------------
 Cumberland St
 Atlantic Commons
```

7. Сколько примерно людей проживает на территории Atlantic Commons (в радиусе 50 метров от нее)?

```
SELECT Sum(popn_total)
  FROM nyc_census_blocks
  WHERE ST_DWithin(
    geom,
```

```
    ST_GeomFromText('LINESTRING(586782
4504202,586864 4504216)', 26918),
    50
  );
```

8. 1438

Пространственные соединения

Пространственные соединения — это основа пространственных баз данных. Они позволяют объединять информацию из разных таблиц, используя пространственные отношения в качестве ключа соединения. Многое из того, что мы называем «стандартным ГИС-анализом», можно выразить как пространственные соединения.
В предыдущем разделе мы исследовали пространственные отношения, используя двухэтапный процесс: сначала мы извлекли точку станции метро «Брод-стрит»; затем мы использовали этот момент, чтобы задать дополнительные вопросы, например: «В каком районе находится станция «Брод-стрит»?» Используя пространственное соединение, мы можем ответить на вопрос за один шаг, получив информацию о станции метро и районе, в котором она находится:

```
SELECT
  subways.name AS subway_name,
  neighborhoods.name AS neighborhood_name,
  neighborhoods.boroname AS borough
FROM nyc_neighborhoods AS neighborhoods
JOIN nyc_subway_stations AS subways
ON ST_Contains(neighborhoods.geom, subways.geom)
WHERE subways.name = 'Broad St';

 subway_name | neighborhood_name   |  borough
-------------+---------------------+-----------
 Broad St    | Financial District | Manhattan
```

Мы могли бы объединить каждую станцию метро с прилегающим к ней районом, но в данном случае нам нужна была информация только об одной. Любая функция, которая обеспечивает связь «истина/ложь» между двумя таблицами, может использоваться для управления пространственным соединением, но наиболее часто используемые из них: ST_Intersects, ST_Contains и ST_DWithin.

Join and Sum

Сочетание JOIN с GROUP BY обеспечивает тот тип анализа, который обычно выполняется в системе ГИС.

Например: «Каково население и расовый состав районов Манхэттена?» Здесь у нас есть вопрос, который объединяет информацию о населении, полученную в результате переписи, с границами кварталов с ограничением только одного района Манхэттена.

```
SELECT
  neighborhoods.name AS neighborhood_name,
  Sum(census.popn_total) AS population,
  100.0 * Sum(census.popn_white) /
Sum(census.popn_total) AS white_pct,
  100.0 * Sum(census.popn_black) /
Sum(census.popn_total) AS black_pct
FROM nyc_neighborhoods AS neighborhoods
JOIN nyc_census_blocks AS census
ON ST_Intersects(neighborhoods.geom, census.geom)
WHERE neighborhoods.boroname = 'Manhattan'
GROUP BY neighborhoods.name
ORDER BY white_pct DESC;

  neighborhood_name   | population | white_pct |
black_pct
```

```
--------------------+------------+-----------+---
--------
 Carnegie Hill      |       18763 |       90.1 |
1.4
 North Sutton Area  |       22460 |       87.6 |
1.6
 West Village       |       26718 |       87.6 |
2.2
 Upper East Side    |      203741 |       85.0 |
2.7
 Soho               |       15436 |       84.6 |
2.2
 Greenwich Village  |       57224 |       82.0 |
2.4
 Central Park       |       46600 |       79.5 |
8.0
 Tribeca            |       20908 |       79.1 |
3.5
 Gramercy           |      104876 |       75.5 |
4.7
 Murray Hill        |       29655 |       75.0 |
2.5
 Chelsea            |       61340 |       74.8 |
6.4
 Upper West Side    |      214761 |       74.6 |
9.2
 Midtown            |       76840 |       72.6 |
5.2
 Battery Park       |       17153 |       71.8 |
3.4
 Financial District |       34807 |       69.9 |
3.8
 Clinton            |       32201 |       65.3 |
7.9
 East Village       |       82266 |       63.3 |
8.8
 Garment District   |       10539 |       55.2 |
7.1
```

```
 Morningside Heights |       42844 |       52.7 |
19.4
 Little Italy       |       12568 |       49.0 |
1.8
 Yorkville          |       58450 |       35.6 |
29.7
 Inwood             |       50047 |       35.2 |
16.8
 Washington Heights |      169013 |       34.9 |
16.8
 Lower East Side    |       96156 |       33.5 |
9.1
 East Harlem        |       60576 |       26.4 |
40.4
 Hamilton Heights   |       67432 |       23.9 |
35.8
 Chinatown          |       16209 |       15.2 |
3.8
 Harlem             |      134955 |       15.1 |
67.1
```

Что тут происходит? Теоретически (фактический порядок вычислений оптимизируется под прикрытием базы данных) происходит вот что:

1. Предложение JOIN создает виртуальную таблицу, включающую столбцы как из таблиц кварталов, так и из таблиц переписи населения.
2. Предложение WHERE фильтрует нашу виртуальную таблицу, оставляя только строки в Манхэттене.
3. Остальные строки группируются по названию района и передаются через функцию агрегирования в Sum() значения населения.
4. После небольшой арифметики и форматирования (например, GROUP BY, ORDER BY) окончательных чисел наш запрос выдает проценты.

Примечание

Предложение JOIN объединяет два элемента FROM. По умолчанию мы используем INNER JOIN, но есть еще четыре типа соединений. Для получения дополнительной информации см. определение join_type в документации PostgreSQL.

Мы также можем использовать дистанционные тесты в качестве ключа соединения для создания обобщенных запросов «все элементы в пределах радиуса». Давайте исследуем расовую географию Нью-Йорка, используя дистанционные запросы.

Во-первых, давайте получим базовый расовый состав города.

```
SELECT
  100.0 * Sum(popn_white) / Sum(popn_total) AS
white_pct,
  100.0 * Sum(popn_black) / Sum(popn_total) AS
black_pct,
  Sum(popn_total) AS popn_total
FROM nyc_census_blocks;
```

```
    white_pct      |    black_pct      | popn_total
-------------------+-------------------+------------
 44.0039500762811 | 25.5465789002416 |    8175032
```

Так, из 8 миллионов жителей Нью-Йорка около 44% записаны как «белые», а 26% — как «черные».

Дюк Эллингтон однажды спел: «Вы / должны сесть на поезд A / Куда / поехать в Шугар-Хилл в Гарлеме». Как мы видели ранее, в Гарлеме, несомненно, самое большое количество афроамериканцев на Манхэттене (80,5%). Верно ли то же самое и с поездом A Дюка?

Во-первых, обратите внимание, что содержимое поля маршрутов таблицы nyc_subway_stations — это то, что нас

интересует, чтобы найти поезд A. Значения там немного сложные.

```
SELECT DISTINCT routes FROM nyc_subway_stations;
A,C,G
4,5
D,F,N,Q
5
E,F
E,J,Z
R,W
```

Note

Ключевое слово DISTINCT исключает повторяющиеся строки из результата. Без ключевого слова DISTINCT приведенный выше запрос выдает 491 результат вместо 73.

Итак, чтобы найти поезд A, нам понадобится любая строка в маршрутах, в которой есть буква «A». Мы можем сделать это несколькими способами, но сегодня мы воспользуемся тем фактом, что strpos(routes,'A') вернет ненулевое число, только если "A" находится в поле routes.

```
SELECT DISTINCT routes
FROM nyc_subway_stations AS subways
WHERE strpos(subways.routes,'A') > 0;

A,B,C
A,C
A
A,C,G
A,C,E,L
A,S
A,C,F
A,B,C,D
A,C,E
```

Давайте подытожим расовый состав людей в радиусе 200 метров от линии поезда A.

```
SELECT
  100.0 * Sum(popn_white) / Sum(popn_total) AS
white_pct,
  100.0 * Sum(popn_black) / Sum(popn_total) AS
black_pct,
  Sum(popn_total) AS popn_total
FROM nyc_census_blocks AS census
JOIN nyc_subway_stations AS subways
ON ST_DWithin(census.geom, subways.geom, 200)
WHERE strpos(subways.routes,'A') > 0;
```

```
    white_pct      |    black_pct      | popn_total
-------------------+-------------------+------------
 45.5901255900202 | 22.0936235670937 |     189824
```

Таким образом, расовый состав жителей поезда A не радикально отличается от расового состава Нью-Йорка в целом.

Упражнения

8. Какая станция метро в «Маленькой Италии»? На какой линии метро?

```
SELECT s.name, s.routes
FROM nyc_subway_stations AS s
JOIN nyc_neighborhoods AS n
ON ST_Contains(n.geom, s.geom)
WHERE n.name = 'Little Italy';
```

```
   name     | routes
------------+--------
```

```
 Spring St | 6
```

9. Какие районы обслуживает поезд №6? (Подсказка: столбец маршрутов в таблице nyc_subway_stations имеет такие значения, как «B,D,6,V» и «C,6»).

```
SELECT DISTINCT n.name, n.boroname
FROM nyc_subway_stations AS s
JOIN nyc_neighborhoods AS n
ON ST_Contains(n.geom, s.geom)
WHERE strpos(s.routes,'6') > 0;
```

```
         name          | boroname
-----------------------+-----------
 Midtown               | Manhattan
 Hunts Point           | The Bronx
 Gramercy              | Manhattan
 Little Italy          | Manhattan
 Financial District    | Manhattan
 South Bronx           | The Bronx
 Yorkville             | Manhattan
 Murray Hill           | Manhattan
 Mott Haven            | The Bronx
 Upper East Side       | Manhattan
 Chinatown             | Manhattan
 East Harlem           | Manhattan
 Greenwich Village     | Manhattan
 Parkchester           | The Bronx
 Soundview             | The Bronx
```

Note

Мы использовали ключевое слово DISTINCT, чтобы удалить повторяющиеся значения из нашего набора результатов, если в районе было более одной станции метро.

10. После событий 11 сентября район «Бэттери-Парк» был закрыт на несколько дней. Сколько человек пришлось эвакуировать?

```
SELECT Sum(popn_total)
FROM nyc_neighborhoods AS n
JOIN nyc_census_blocks AS c
ON ST_Intersects(n.geom, c.geom)
WHERE n.name = 'Battery Park';

17153
```

11. В каком районе самая высокая плотность населения (чел./км2)?

```
SELECT
  n.name,
  Sum(c.popn_total) / (ST_Area(n.geom) /
1000000.0) AS popn_per_sqkm
FROM nyc_census_blocks AS c
JOIN nyc_neighborhoods AS n
ON ST_Intersects(c.geom, n.geom)
GROUP BY n.name, n.geom
ORDER BY popn_per_sqkm DESC LIMIT 2;

      name        |  popn_per_sqkm
------------------+------------------
 North Sutton Area | 68435.13283772678
 East Village      | 50404.48341332535
```

Есть несколько распространенных типов геопространственных данных, с которыми вам необходимо ознакомиться: шейп-файл (.shp) и GeoJSON (.geojson).

- How many census blocks don’t contain their own centroid?

```
SELECT Count(*)
  FROM nyc_census_blocks
  WHERE NOT
    ST_Contains(
      geom,
      ST_Centroid(geom)
    );
481
```

- Union all the census blocks into a single output. What kind of geometry is it? How many parts does it have?

```
CREATE TABLE nyc_census_blocks_merge AS
  SELECT ST_Union(geom) AS geom
  FROM nyc_census_blocks;

SELECT ST_GeometryType(geom)
  FROM nyc_census_blocks_merge;
ST_MultiPolygon
SELECT ST_NumGeometries(geom)
  FROM nyc_census_blocks_merge;
63
```

- What is the area of a one unit buffer around the origin? How different is it from what you would expect? Why?

```
SELECT ST_Area(ST_Buffer('POINT(0 0)', 1));
3.121445152258052
```

Note

A unit circle (circle with radius of one) should have an area of pi, 3.1415926… The difference is due to the linear stroking of the edges of the buffer. The buffer has a finite number of edges. Increasing the number of edges in the buffer will get the value closer to pi, but it will always be smaller due to the linearization.

- The Brooklyn neighborhoods of ‘Park Slope’ and ‘Carroll Gardens’ are going to war! Construct a polygon

delineating a 100 meter wide DMZ on the border between the neighborhoods. What is the area of the DMZ?

```
CREATE TABLE brooklyn_dmz AS
  SELECT
    ST_Intersection(
      ST_Buffer(ps.geom, 50),
      ST_Buffer(cg.geom, 50))
    AS geom
  FROM
    nyc_neighborhoods ps,
    nyc_neighborhoods cg
  WHERE ps.name = 'Park Slope'
  AND cg.name = 'Carroll Gardens';

SELECT ST_Area(geom) FROM brooklyn_dmz;
```

Note

It is easy to buffer both the neighborhoods of interest, but to get the intersection requires a self-join of the table, creating one relation (`ps`) with just the “Park Slope” record and another (`cg`) with just the “Carroll Gardens” record. Note that the area of the intersection is in square meters because we are still working in UTM 18 (EPSG:26918).

```
180990.964207547
```

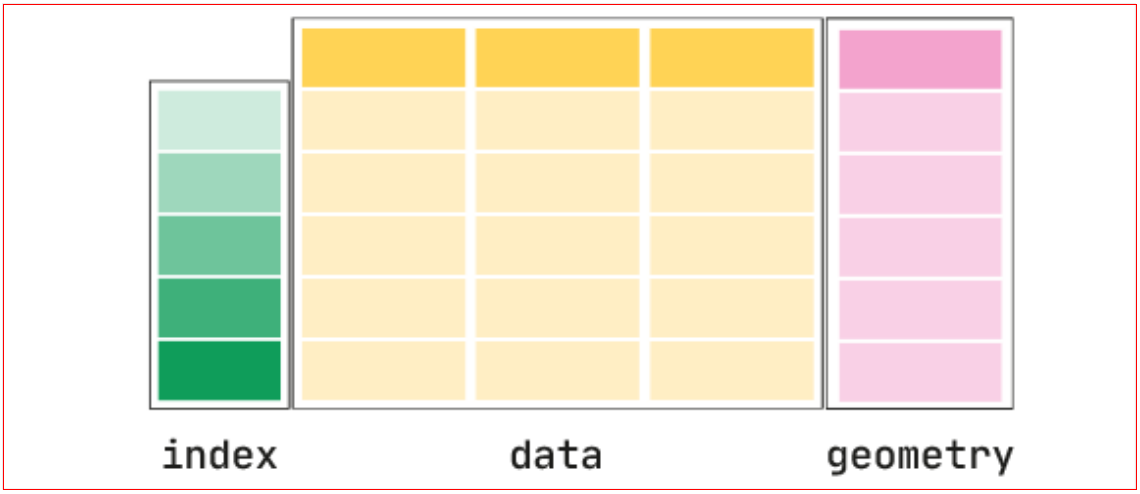
Shapefile - это формат векторных данных, который в основном разрабатывается и поддерживается компанией ESRI. В нем хранится много важной геопространственной информации, включая топологию, геометрию формы и т. Д.

GeoJSON, как и JSON, хранит
информацию геометрии (координаты, проекция и т. д.) в дополнение к типичным атрибутам, относящимся к объекту (индекс, имя и т. д.).

GeoJSON

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
  },
  "properties": {
    "name": "Dinagat Islands"
  }
}
```

После загрузки любого из этих форматов данных с помощью Geopandas библиотека создаст DataFrame с дополнительным столбцом geometry.



Создание пространственной базы данных на языке Python с использованием библиотеки GeoPandas позволяет работать с геоданными в Python-среде, что очень удобно для анализа, обработки и визуализации данных. GeoPandas предоставляет множество инструментов для работы с пространственными данными, в том числе возможность загрузки, обработки и сохранения геоданных в различных форматах, а также гибкие инструменты для манипулирования геометрией объектов и атрибутами данных.

Создание пространственной базы данных на языке Python позволяет:

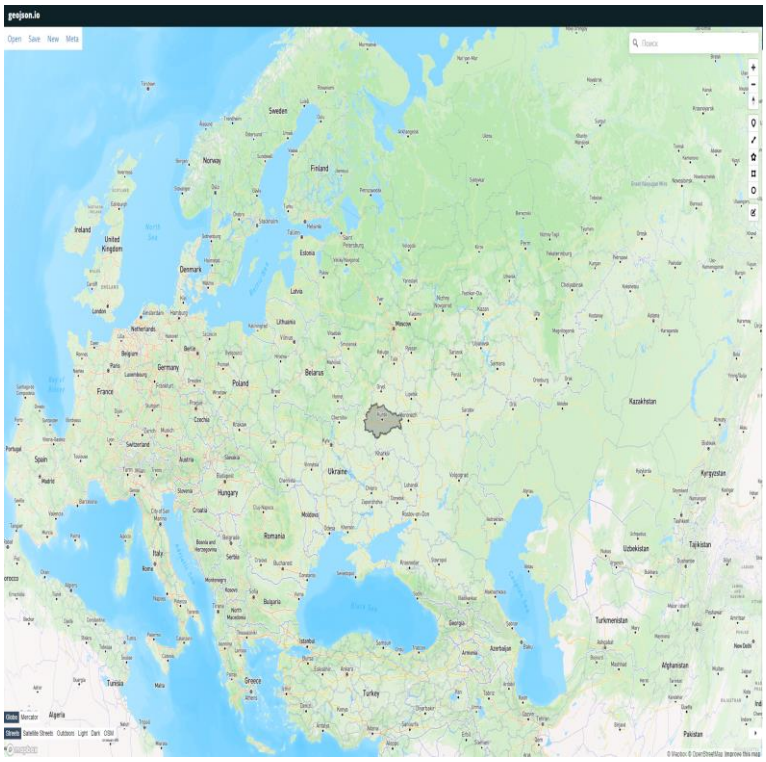
9. Создавать и хранить геопространственные данные
10. Агрегировать данные по регионам или другим географическим областям

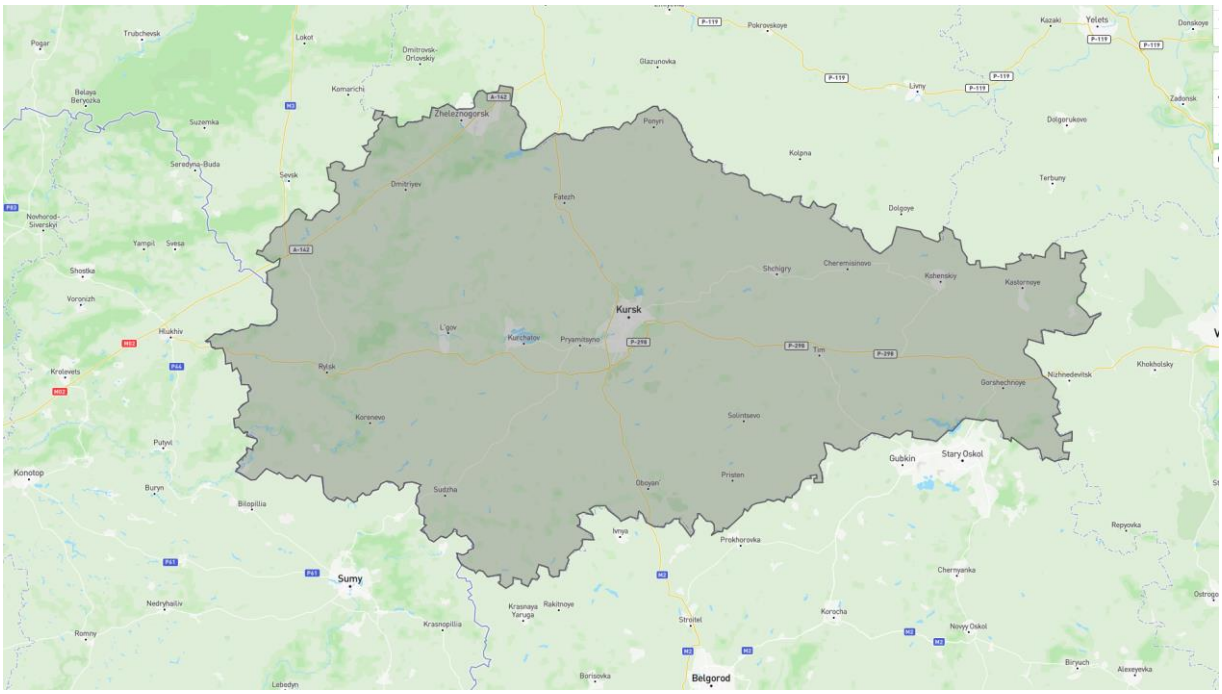
11. Производить пространственный анализ данных, например, определять расстояния между объектами, строить тепловые карты, находить ближайшие объекты, определять, какие объекты находятся внутри заданных границ

12. Сохранять и загружать геопространственные данные из различных форматов, например, для обмена данными с другими системами.

13. Визуализировать геопространственные данные на карте

Вот пример визуализации Курской области





В этой работе мы научимся создавать пространственную базу данных, которая будет содержать информацию о городах, их населении, регионе, населении региона, геометрии городов (в виде простого полигона) и центроидах (среднее арифметическое положение всех точек фигуры) городов. Далее мы соединим несколько полигонов линиями для визуального отображения. После этого мы сохраним базу данных в файл формата .geojson, для дальнейшего визуального отображения проделанной работы.

12. Импортируем необходимые библиотеки:

```
import geopandas as gpd
from matplotlib import pyplot as plt
from shapely import Point
from shapely.geometry import Polygon, LineString
```

13. В данном пункте мы создаем базу данных, которая будет содержать информацию о городах, их населении, регионе, населении региона, геометрии городов (в виде полигона) и центроидах городов. Для этого мы используем словарь data.

Словарь data содержит следующие ключи:

a. 'city': список названий городов,
b. 'population': список населения городов,
c. 'geometry': список полигонов, описывающих геометрию городов,
d. 'region': список названий регионов, в которых находятся города,
e. 'region_population': список населения регионов,
f. 'center': список координат центроидов городов.

Мы создаем GeoDataFrame gdf, который будет содержать все эти данные. В качестве аргументов мы передаем наш словарь data, указываем координатную систему crs='EPSG:4326' и геометрию, которая будет описывать полигоны нашего GeoDataFrame, geometry='geometry'.

Таким образом, код для создания GeoDataFrame будет выглядеть следующим образом:

```
    data = {'city': ['Москва', 'Санкт-Петербург', 'Новосибирск'],
            'population': [12678079, 5384342, 1625631],
            'geometry': [Polygon([(37.4166, 55.7999), (37.4166, 56.0000), (37.8666, 56.0000), (37.8666, 55.7999)]) ,
                         Polygon([(30.1298, 59.8219), (30.1298, 60.0917), (30.4846, 60.0917), (30.4846, 59.8219)]) ,
                         Polygon([(82.7116, 54.6623), (82.7116, 55.0375), (83.3463, 55.0375), (83.3463, 54.6623)]) ],
            'region': ['Московская область', 'Ленинградская область', 'Новосибирская область'],
            'region_population': [7471000, 1806000, 2700000],
            'center': [(37.6175, 55.7558), (30.3351, 59.9343), (82.9346, 55.0302)]}

    gdf = gpd.GeoDataFrame(data, crs='EPSG:4326', geometry='geometry')
```

14. Создаем GeoDataFrame на основе этой базы данных. Параметр crs определяет систему координат для этого GeoDataFrame (EPSG:4326 — это WGS84, широко используемая система координат для географических данных).

```
gdf = gpd.GeoDataFrame(data, crs='EPSG:4326', geometry='geometry')
```

15. Получаем центроиды каждого полигона внутри MultiPolygon, чтобы затем соединить города линиями.

```
centroids = [Point(c) for c in gdf['center']]
```

16. Создаем геометрию линии, соединяющую города. Для этого мы извлекаем координаты центроидов каждого города и создаем LineString из этих координат.

```
line = LineString([c.coords[0] for c in centroids])
```

17. Создаем GeoDataFrame с геометрией линии

```
line_gdf = gpd.GeoDataFrame(geometry=[line], crs='EPSG:4326')
```

Здесь мы создаем новый GeoDataFrame line_gdf с единственным объектом геометрии - линией, которая соединяет центроиды всех городов. Эта линия будет использоваться для визуализации на карте в дальнейшем.

18. Объединяем геометрии в одну общую геометрию

```
geom_union = gdf['geometry'].unary_union
```

Здесь мы используем метод unary_union для объединения геометрий всех городов в одну общую геометрию geom_union. Это необходимо для того, чтобы мы могли определить область, которую нужно визуализировать на карте.

19. Создаем GeoSeries с геометриями городов

```
cities = gdf['geometry']
```

20. Получаем центроиды каждого полигона внутри MultiPolygon

```
centroids = [p.centroid for p in cities]
```

21. Создаем GeoDataFrame с линией и геометриями городов

```
combined_gdf = gpd.GeoDataFrame(geometry=list(cities) + [line], crs='EPSG:4326')
```

Здесь мы создаем новый GeoDataFrame combined_gdf, включающий в себя геометрии городов и линию, соединяющую их. Мы также добавляем данные о городах, которые мы сохранили в исходном GeoDataFrame gdf.

22. Добавляем данные о городах в новый GeoDataFrame

```
combined_gdf['city'] = gdf['city']
combined_gdf['population'] = gdf['population']
combined_gdf['region'] = gdf['region']
combined_gdf['region_population'] = gdf['region_population']
combined_gdf = combined_gdf[['city', 'population', 'region', 'region_population', 'geometry']]
```

Здесь мы добавляем данные о городах в новый GeoDataFrame combined_gdf, которые мы сохранили в исходном GeoDataFrame gdf. Затем мы переупорядочиваем столбцы, чтобы сначала шли данные о городах, а затем геометрии.

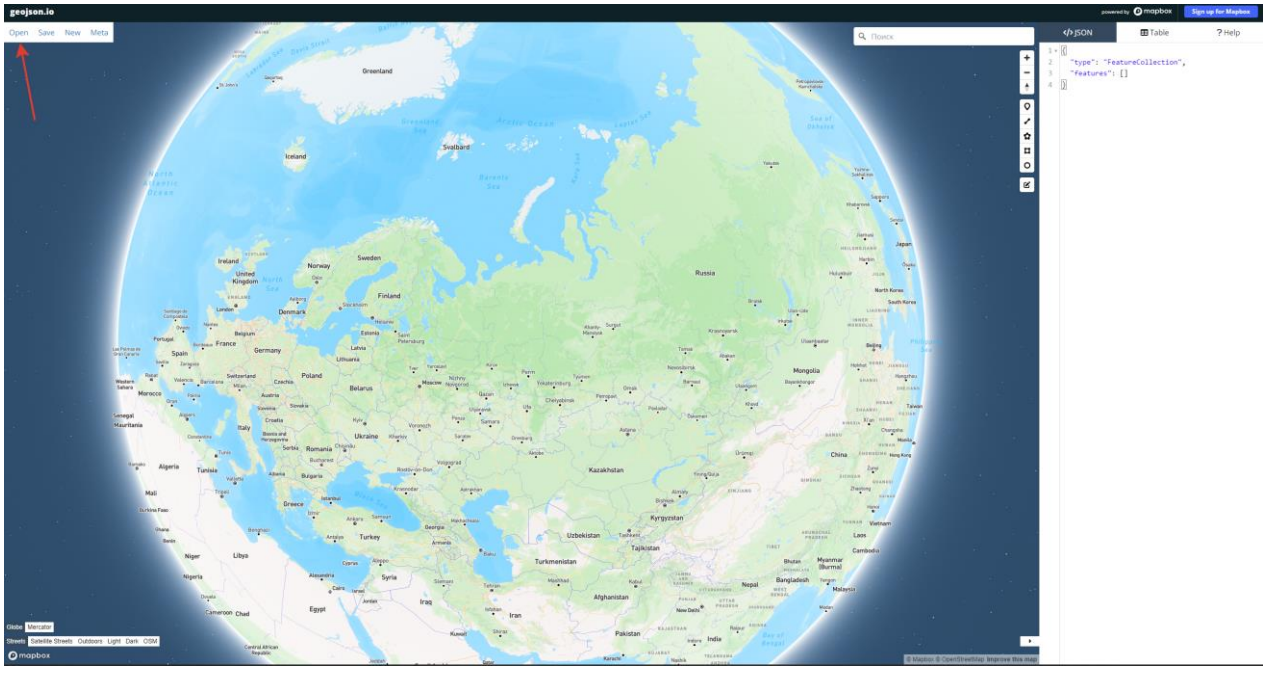
23. Сохраняем GeoDataFrame в файл GeoJSON

```
combined_gdf.to_file('cities_and_line.geojson', driver='GeoJSON', driveroptions={'keep_infinity': True})
```

24. Рисуем карту. Мы используем метод plot для каждого GeoDataFrame и передаем ось рисунка в качестве параметра. В результате получаем карту, на которой показаны границы городов и линия, соединяющая эти города.

```
fig, ax = plt.subplots()
combined_gdf.plot(ax=ax)
line_gdf.plot(ax=ax, color='red')
plt.show()
```

25. Открываем сайт https://geojson.io и загружаем наш сгенированный geojson файл



26. Проверяем чтобы созданные полигоны сходились с базой данных, также проверяем отрисованные линии