Introduction to PostGIS and using PostGIS from R

What is PostGIS

- Adds support for geographic objects to the PostgreSQL object-relational database

- PostgreSQL already has “geometric types” but native geometries are too limited for GIS data and analysis

What is PostGIS

- PostGIS adds an indexing mechanism to allow queries with spatial restrictions or filters (e.g., “tuples within this bounding box”) to return records very quickly from large tables.

What is PostGIS?

- PostGIS adds a “geometry” data type to the usual data types of the relational database (ie. varchar, integer, date, etc.)

- PostGIS adds spatial predicates and functions using the geometry data type.
  - ST_Distance(geometry, geometry)
  - ST_Area(geometry)
  - ST_Intersects(geometry, geometry))

Geometric Datatypes

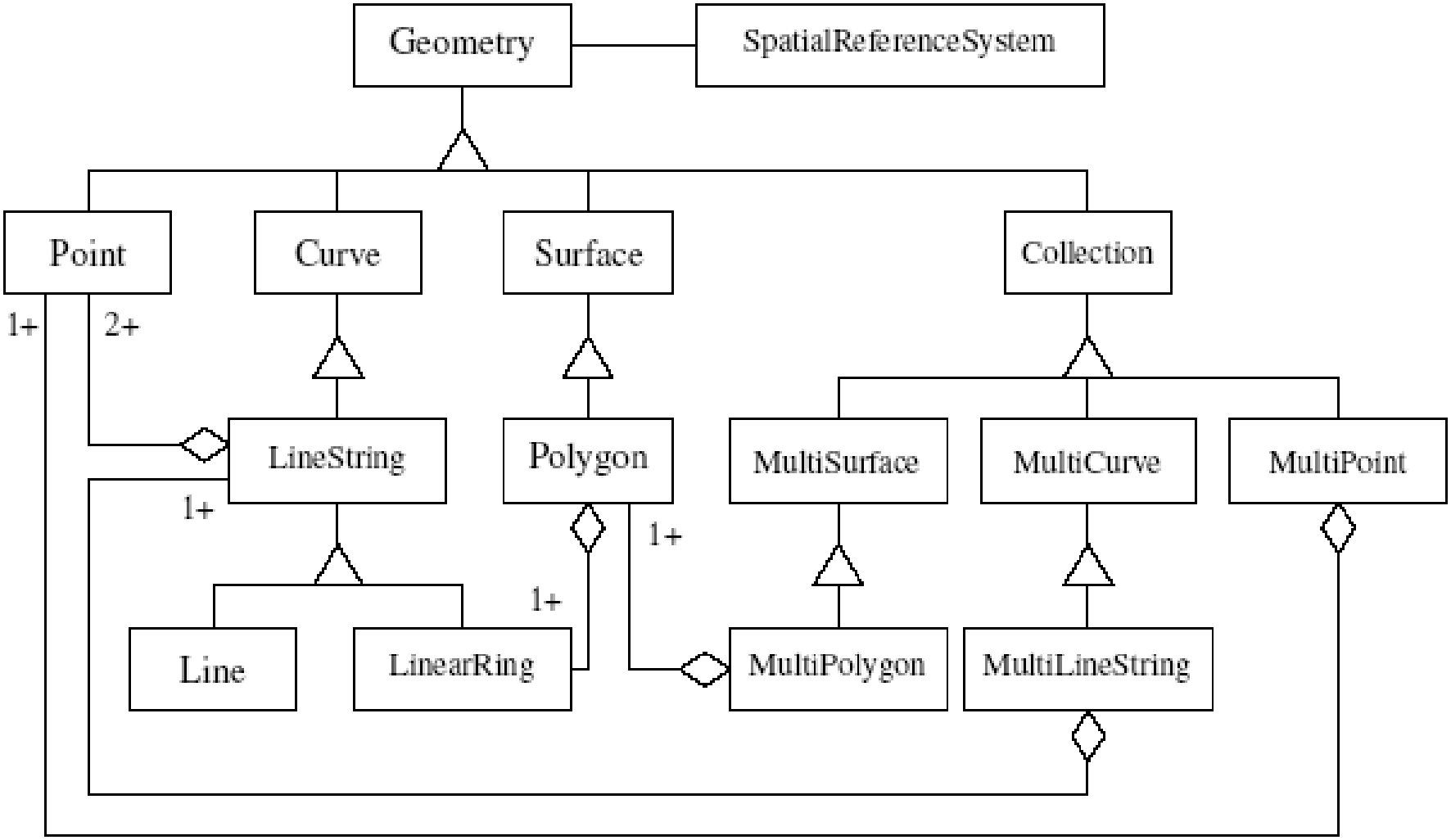


Figure 1: OpenGIS Geometry Class Hierarchy

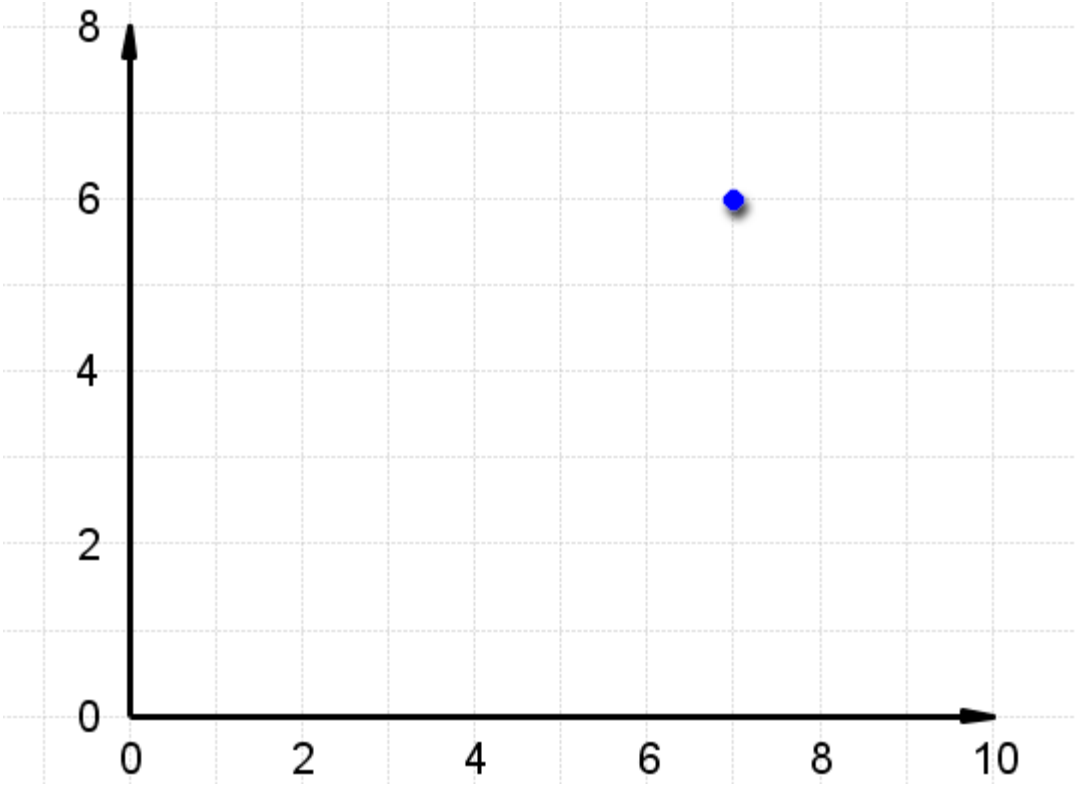
Geometric Datatypes

- Each geometry has a Well-Known Text representation (WKT)
  - A geometry type
  - A comma-separated list of coordinate pairs
- Internal binary representation
- Serialization to Geography Markup Language (GML) geometry objects

Geometric Datatypes

- Some examples are:

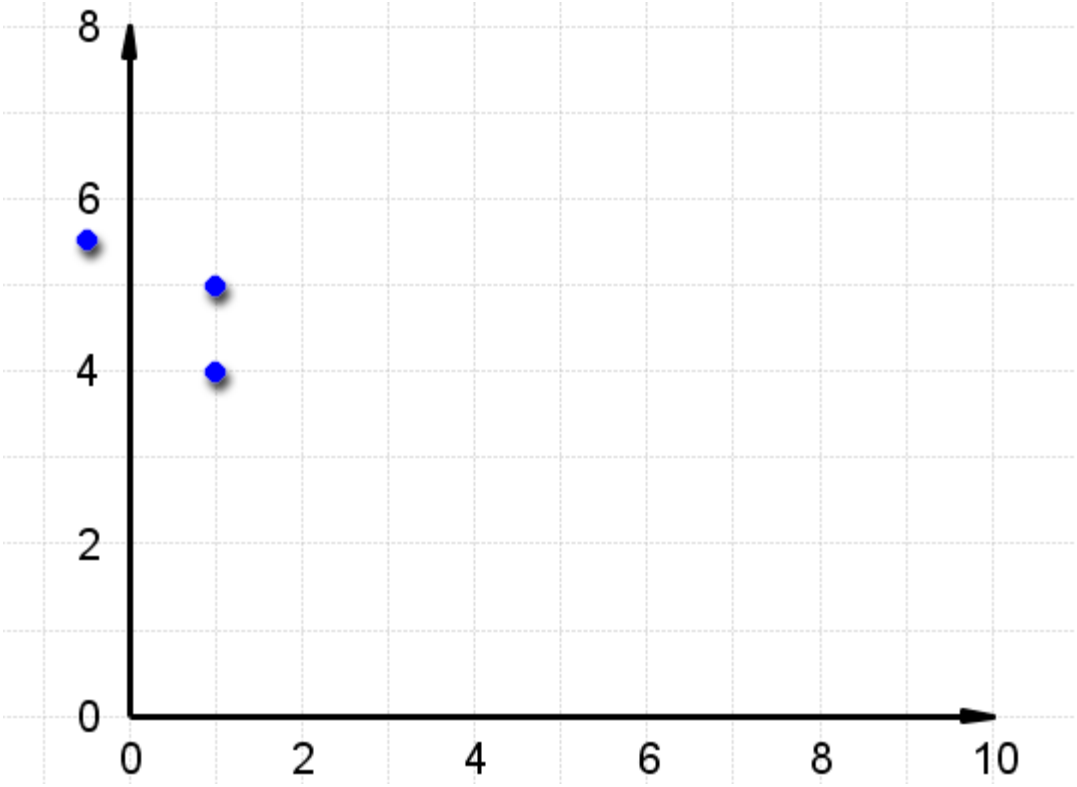
`POINT ( 7 6 )`



Geometric Datatypes

- Some examples are:

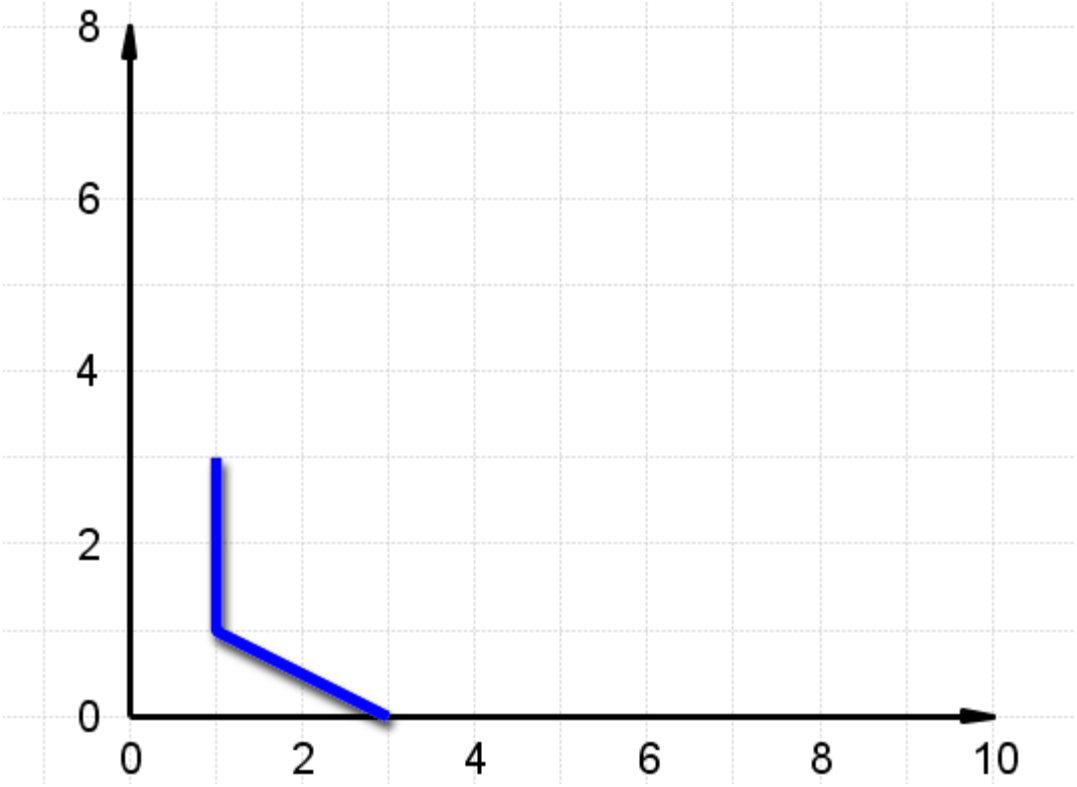
```
MULTIPOINT ( 1 4, 1 5, -0.5 5.5314 )
```



Geometric Datatypes

- Some examples are:

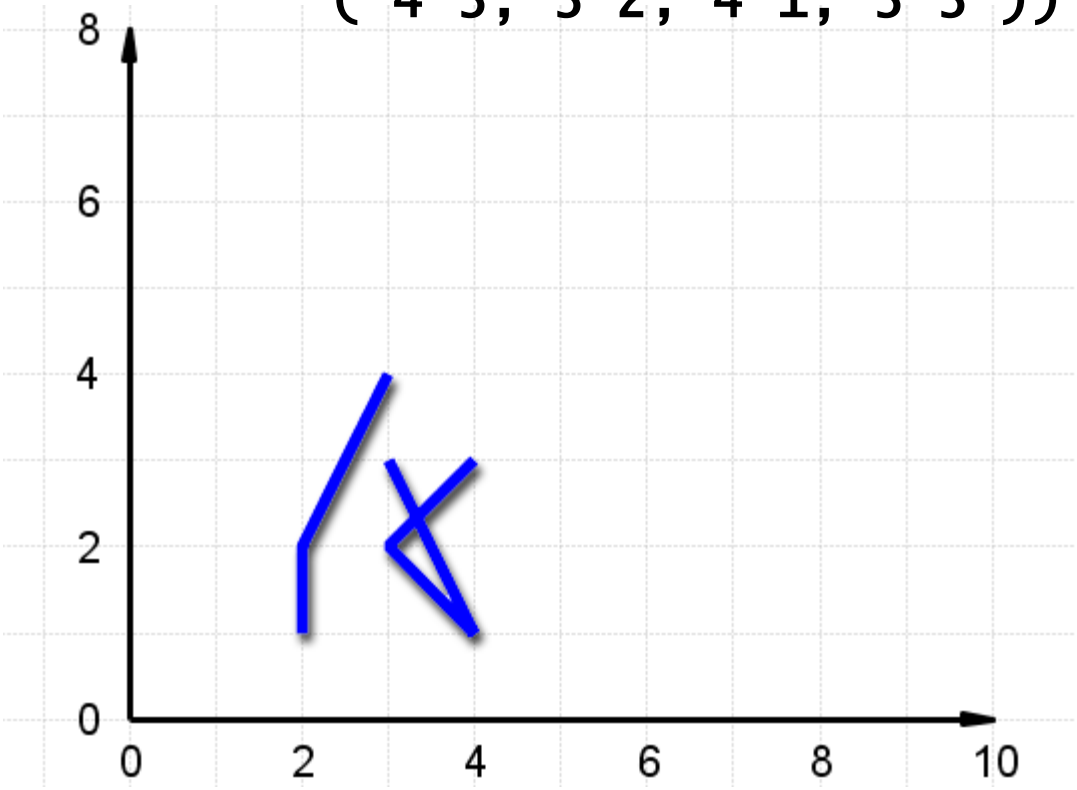
```
LINESTRING ( 1 3, 1 1, 3 0 )
```



Geometric Datatypes

- Some examples are:

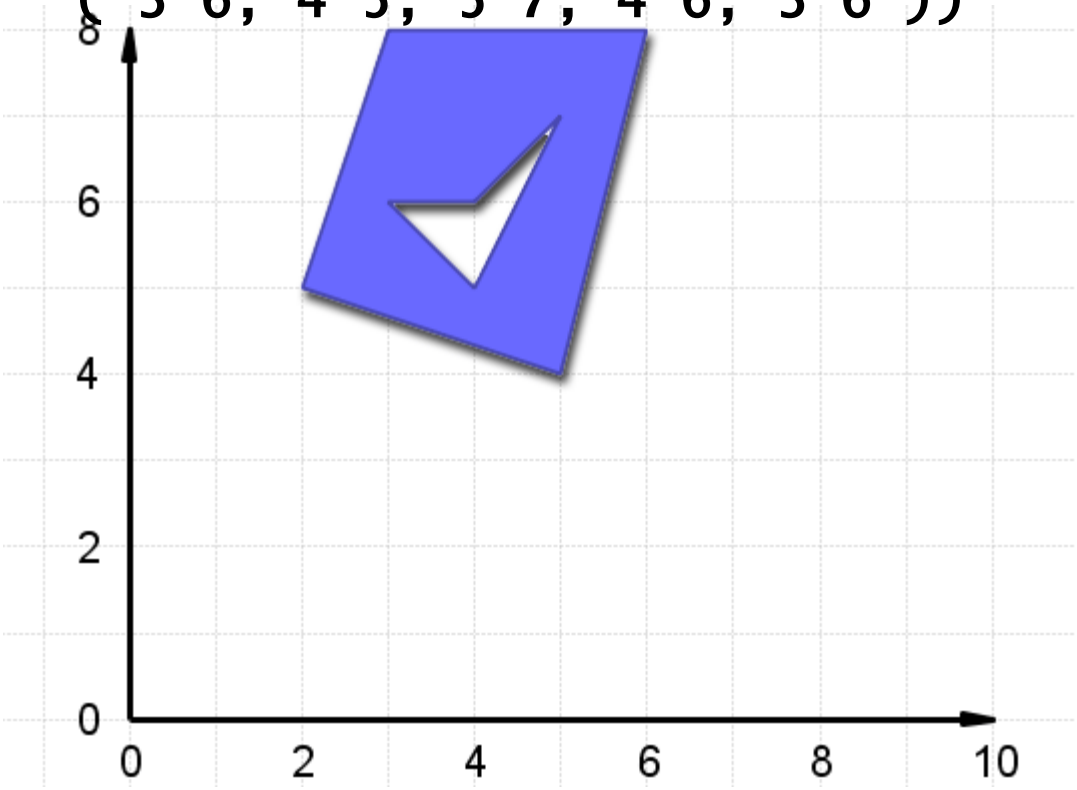
```
MULTILINESTRING (( 2 1, 2 2, 3 4 ),
                 ( 4 3, 3 2, 4 1, 3 3 ))
```



Geometric Datatypes

- Some examples are:

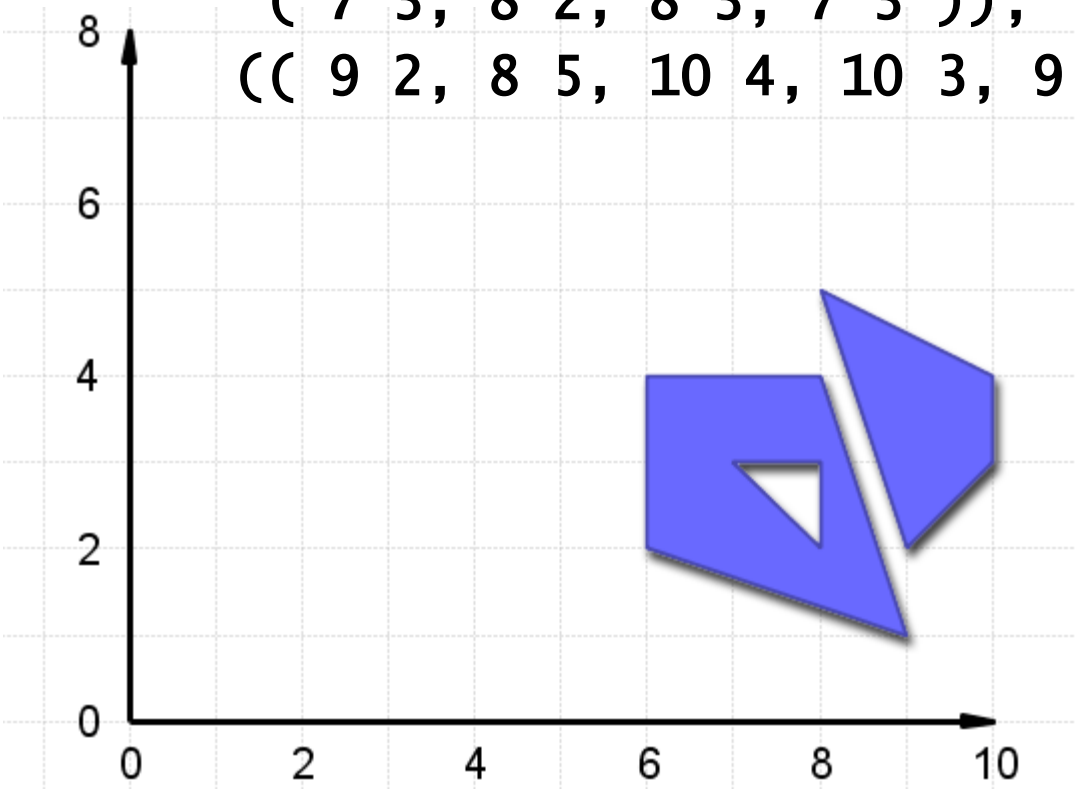
```
POLYGON (( 2 5, 3 8, 6 8, 5 4, 2 5 ),
         ( 3 6, 4 5, 5 7, 4 6, 3 6 ))
```



Geometric Datatypes

- Some examples are:

```
MULTIPOLYGON ((( 6 2, 6 4, 8 4, 9 1, 6 2 ),
                ( 7 3, 8 2, 8 3, 7 3 )),
              (( 9 2, 8 5, 10 4, 10 3, 9 2 )))
```



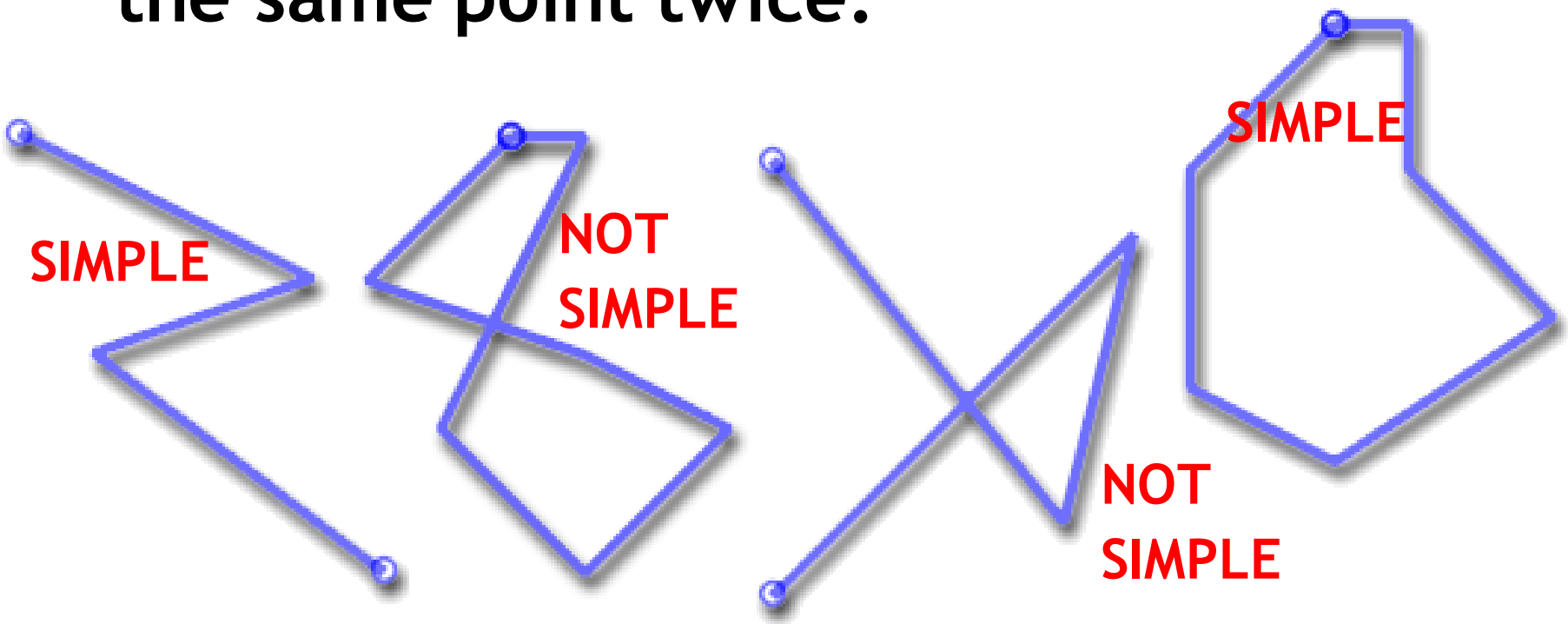
2.3 - Geometry Validity

- PostGIS is compliant with the Open Geospatial Consortium's (OGC) OpenGIS Specifications

- Functions require/assume geometries are valid and in many cases, simple.
  - Geometry validity really only applies for areal geometries
  - Geometry simplicity applies to point and linear geometies

Geometry Validity and Simplicity

- By definition, a `LINESTRING` is always *valid*
- It is *simple* if it does not pass through the same point twice.



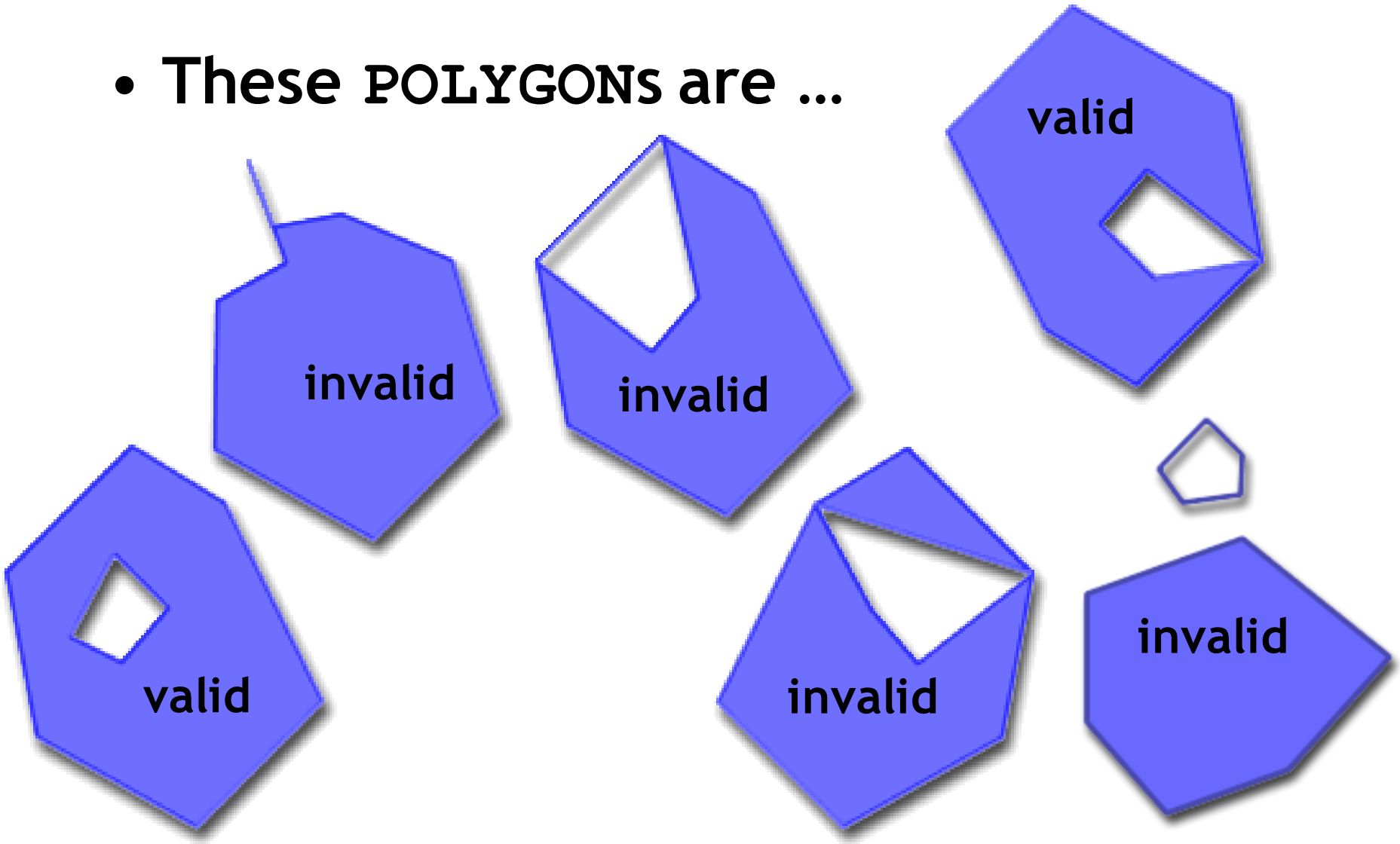
Geometry Validity

- By definition, a `POLYGON` is always *simple*.

- It is *valid* if
  - The boundary is made up of *simple* `LINESTRING`s
  - boundary rings do not cross
  - holes are completely within the outer ring and touch the outer ring at most one point.
  - it does not contain cutlines / spikes

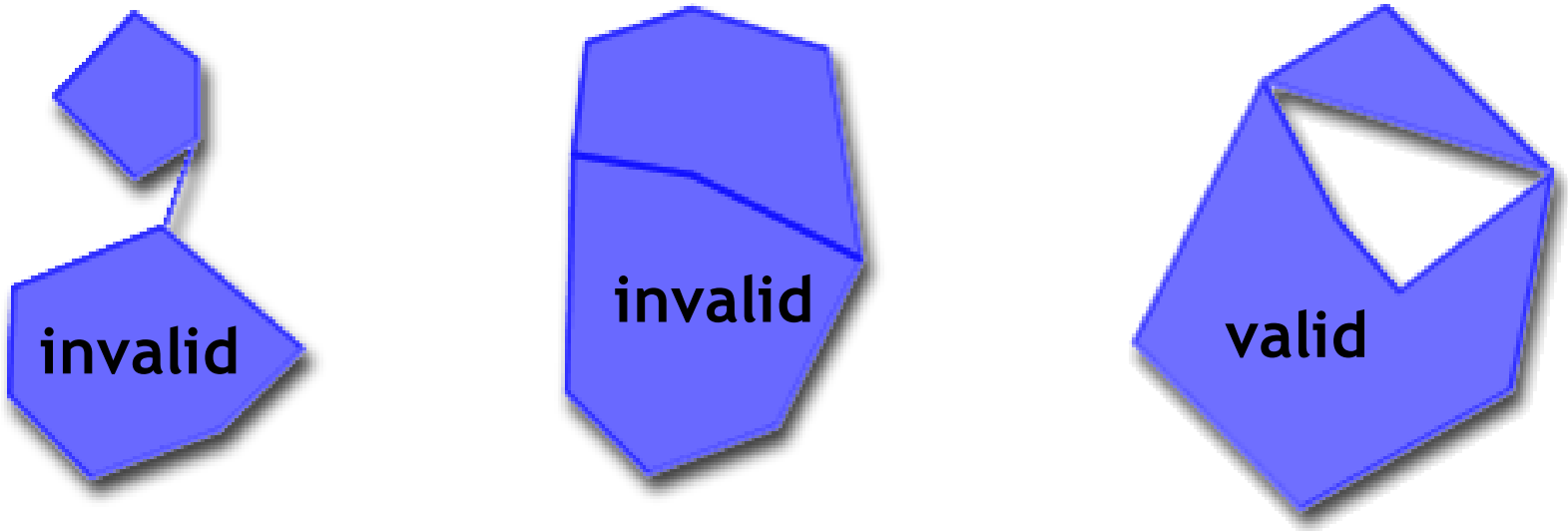
Geometry Validity

- These `POLYGONs` are …



Geometry Validity

- By definition, a `MULTIPOLYGON` is *valid* iff
  - All elements are valid
  - Element interiors cannot intersect
  - Element boundaries can touch, but only at a finite number of `POINT`s.



PostgreSQL + PostGIS Installation

- **Windows Installer**
  - **PostgreSQL + PostGIS**
    - **Installs itself as a Windows Service**
  - **PgAdmin III**
    - **Psql Interactive SQL Shell**
  - **http://postgis.refractions.net/download/windows/**

- **Linux Instaler**
  - sudo **apt-get install** postgresql-8.2-postgi pgadmin3
  - http://postgis.refractions.net/docs/ch02.html

Create Spatial Database

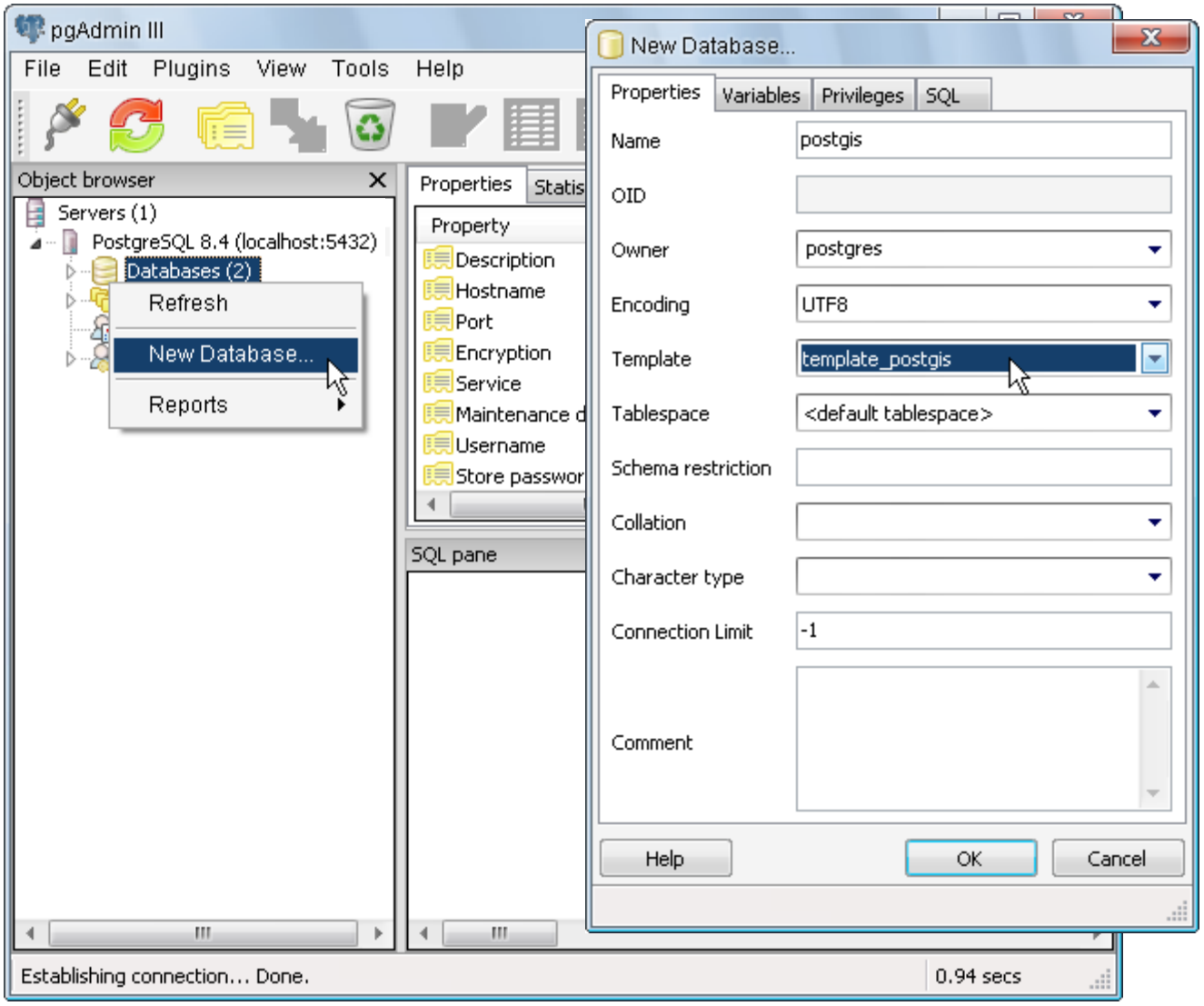
- From the command line

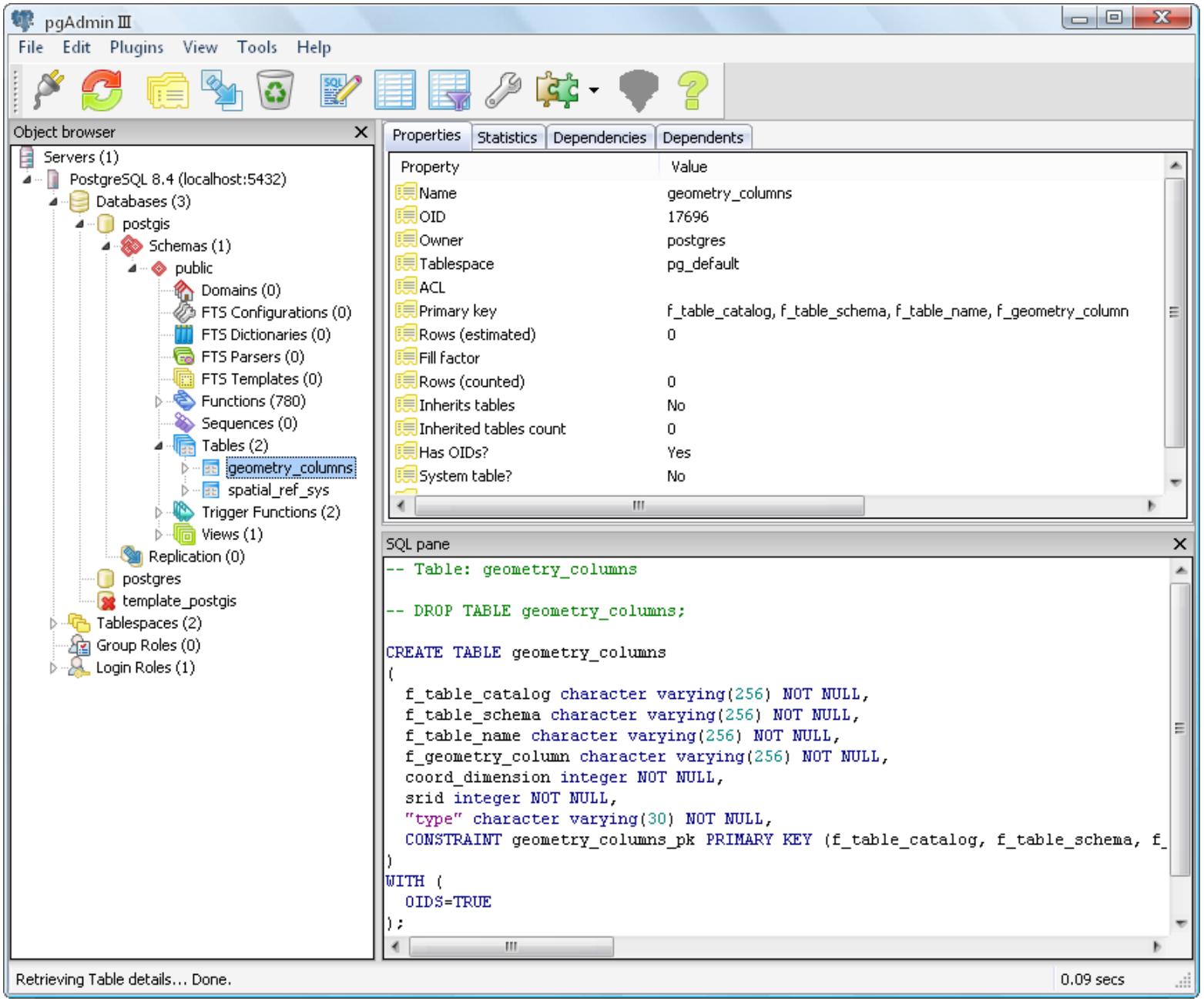
```
createdb -T template_postgis dbname
```

- Create a new database
  - Select “template_postgis” as the template

- Verify the installation
  - Check for the spatial system tables
    - *spatial_ref_sys* - stores unique coordinate references
    - *geometry_columns* - stores metadata for a spatial column

Create Spatial Database




pgAdmin III
File Edit Plugins View Tools Help
Object browser
Servers (1)
PostgreSQL 8.4 (localhost:5432)
Databases (3)
postgis
Schemas (1)
public
Domains (0)
FTS Configurations (0)
FTS Dictionaries (0)
FTS Parsers (0)
FTS Templates (0)
Functions (780)
Sequences (0)
Tables (2)
geometry_columns
spatial_ref_sys
Trigger Functions (2)
Views (1)
Replication (0)
postgres
template_postgis
Tablespaces (2)
Group Roles (0)
Login Roles (1)
Properties Statistics Dependencies Dependents
Property Value
Name geometry_columns
OID 17696
Owner postgres
Tablespace pg_default
ACL
Primary key f_table_catalog, f_table_schema, f_table_name, f_geometry_column
Rows (estimated) 0
Fill factor
Rows (counted) 0
Inherits tables No
Inherited tables count 0
Has OIDs? Yes
System table? No
SQL pane
-- Table: geometry_columns
-- DROP TABLE geometry_columns;
CREATE TABLE geometry_columns
(
f_table_catalog character varying(256) NOT NULL,
f_table_schema character varying(256) NOT NULL,
f_table_name character varying(256) NOT NULL,
f_geometry_column character varying(256) NOT NULL,
coord_dimension integer NOT NULL,
srid integer NOT NULL,
"type" character varying(30) NOT NULL,
CONSTRAINT geometry_columns_pk PRIMARY KEY (f_table_catalog, f_table_schema, f_
)
WITH (
OIDS=TRUE
);
Retrieving Table details... Done.
0.09 secs

Spatially-Enable an Existing DB

- From the command line

```
psql -f "%PGHOME%\share\contrib\postgis-1.5\postgis.sql" dbname
psql -f "%PGHOME%\share\contrib\postgis-1.5\spatial_ref_sys.sql" dbname
```

- Using a User Interface
  - Connect to your existing database
  - Open the “Execute Arbitrary SQL Queries” window
  - Load/run the PostGIS extension (postgis.sql)
  - Load/run the PostGIS spatial reference systems (spatial_ref_sys.sql)

Simple Spatial SQL

- "Manually" create geometries

```
CREATE TABLE points
   (pt geometry, name varchar);
INSERT INTO points VALUES
   ('POINT(0 0)', 'Origin');
INSERT INTO points VALUES
   ('POINT(5 0)', 'X Axis');
INSERT INTO points VALUES
   ('POINT(0 5)', 'Y Axis');

SELECT name, ST_AsText(pt),
         ST_Distance(pt, 'POINT(5 5)')
FROM points;
```

PostGIS Reference

- Function References:

http://postgis.refractions.net/docs/ch07.html

Loading Shape Files

- Shape File (Misnomer! 3+ Files!)
  - .shp = geometry
  - .dbf = attributes (string, number, date)
  - .shx = utility index
- PostGIS/PostgreSQL Table
  - Columns can be geometry
  - Columns can be attributes
- One Shape File = One PostGIS Table

Loading Shape Files

- shp2pgsql [opts] shapefile tablename
  - shp2pgsql -i -s 3005 -D bc_pubs.shp bc_pubs > bc_data.sql
- Read in .shp file
- Write out .sql file
- Load .sql file into PostgreSQL
  - using psql
  - using PgAdmin

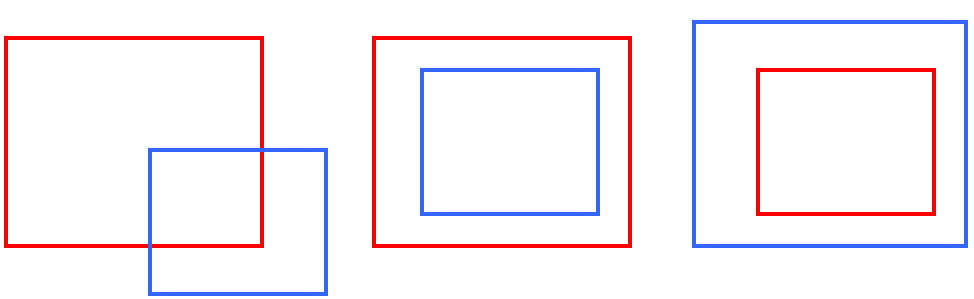
Creating Spatial Indexes

- An example index creation command is

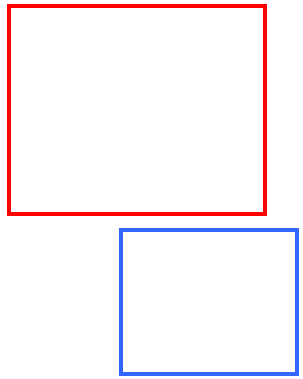
```
CREATE INDEX points_gidx ON points
   USING GIST (pt);
```

Using Spatial Indexes

- Spatial index operator is "&&"
  - "Bounding Boxes Interact"



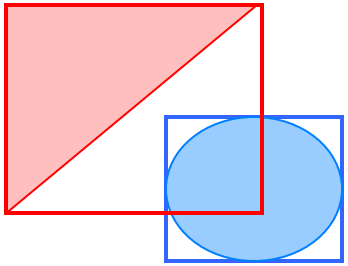
A && B = TRUE



A && B = FALSE

Using Spatial Indexes

- Bounding Boxes are not enough!



A && B = TRUE
_ST_Intersects(A, B) = FALSE

- Two pass system required
  - Use bounding boxes to reduce candidates
  - Use real topological test to get final answer

Using Spatial Indexes

- Index operations (&&) are built into common functions for automatic use, but you can use them separately if you like.
  - ST_Intersects(G1,G2)
    - G1 && G2 AND _ST_Intersects(G1,G2)
  - ST_Contains(G1,G2)
  - ST_Within(G1,G2)
  - ST_Touches(G1,G2)
  - ST_DWithin(G1,G2,D)
    - G1 && ST_Expand(G2,D) AND ST_Distance(G1,G2) > D

Spatial Analysis in SQL

- A surprising number of traditional *GIS analysis* questions can be answered using a spatial database and SQL.

- *GIS analysis* is generally about filtering spatial objects with conditions, and summarizing the results – and that is exactly what databases are very good at.

Distance Queries

- It’s easy to write inefficient queries for distances in PostGIS because it’s not always obvious how to use the index in a distance query.

Distance Queries

Question: **How many wildlife habitats are within 20 kilometers of the municipality of Oliver?**

First, find out where Oliver is located.

```
SELECT ST_AsText(ST_Centroid(the_geom))
FROM bc_municipalities
WHERE name = 'Oliver';

POINT(1470065.29710885 484600.794443135)
```

Distance Queries

Then, use that location to sum all habitats within 20km. The "obvious" way does not use any indexes.

```
SELECT count(*) AS num_habitats
FROM bc_habitat_areas
WHERE
  ST_Distance(
    the_geom,
    ST_GeomFromText('POINT(1470065 484600)',3005)
  ) < 20000;
```

Distance Queries

The "optimized" way is to use `ST_DWithin()` which silently adds an index into the mix.

```
SELECT count(*) AS num_habitats
FROM bc_habitat_areas
WHERE
  ST_DWithin(
    the_geom,
    ST_GeomFromText(
      'POINT(1470065 484600)',3005),
      20000
  );
```

Distance Queries

Here's the definition of `ST_DWithin()`.

```
CREATE FUNCTION ST_DWithin(geometry, geometry,
float8)
  RETURNS boolean AS '
    SELECT
      $1 && ST_Expand($2,$3) AND
      $2 && ST_Expand($1,$3) AND
      ST_Distance($1, $2) < $3
  ' LANGUAGE 'SQL' IMMUTABLE;
```

Spatial Joins

- **Standard joins use a common key**

```
SELECT a.var1, b.var2
FROM a, b
WHERE a.id = b.id
```

- **Spatial joins are based on a spatial relationship**

```
SELECT a.var1, b.var2
FROM a, b
WHERE ST_Intersects(a.geom, b.geom)
```

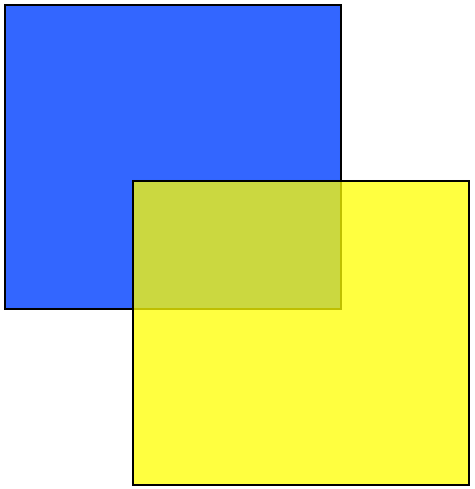
Spatial Joins

- **The previous distance query can be expressed using a single spatial join.**

```
SELECT count(*) AS num_habitats
FROM
  bc_municipalities a,
  bc_habitat_areas b
WHERE
  ST_DWithin(
    a.the_geom,
    b.the_geom,
    20000)
AND
  a.name ILIKE 'Oliver%';
```

Overlays

- Table-on-table overlays are possible with the ST_Intersection() function
  - ST_Intersects(a,b) returns BOOLEAN
  - ST_Intersection(a,b) returns GEOMETRY



ST_Intersects() = TRUE

ST_Intersection() = 

Overlays

Question: **Create a new table of habitat areas clipped by the Chilliwack municipal boundary.**

Overlays

```
CREATE TABLE ch_habitat_areas AS
SELECT
      ST_Intersection(h.the_geom, m.the_geom)
            AS intersection_geom,
      ST_Area(h.the_geom) AS va_area,
      h.*,
      m.name AS municipality_name
FROM
      bc_habitat_areas h,
      bc_municipalities m
WHERE ST_Intersects(h.the_geom, m.the_geom)
AND m.name = 'Chilliwack';
```

Coordinate Projection

- Every geometry in PostGIS has a “spatial referencing identifier” or “SRID” attached to it.

- The SRID indicates the spatial reference system the geometry coordinates are in.

Coordinate Projection

- View the SRID of geometries using the ST_SRID() function

```
SELECT ST_SRID(the_geom)
FROM bc_roads LIMIT 1;
```

3005

- What's 3005?

```
SELECT srtext
FROM spatial_ref_sys
WHERE srid = 3005;
```

PROJCS["NAD83 / BC Albers",…

- Ah, it's "BC Albers"

Coordinate Projection

- **What's 3005 again?**

```
SELECT proj4text
FROM spatial_ref_sys
WHERE srid = 3005;
```

```
"+proj=aea +lat_1=50 +lat_2=58.5
  +lat_0=45 +lon_0=-126 +x_0=1000000
  +y_0=0 +ellps=GRS80 +datum=NAD83
  +units=m +no_defs"
```

- **PROJ4 is coordinate re-projection library used by PostGIS**

Coordinate Projection

- Coordinate Re-projection is done using the ST_Transform() function

```
SELECT ST_AsEWKT(the_geom)
FROM bc_roads
LIMIT 1;
"SRID=3005;MULTILINESTRING((
  1004687.04355194 594291.053764096,
  1004729.74799931 594258.821943696,
  1004808.0184134 594223.285878035,
  1004864.93630072 594204.422638658,
  1004900.50302171 594200.005856311
))"
```

Extracting Spatial Data

- *One of the utility applications that accompany a typical PostGIS installation is the* `pgsql2shp` *command-line utility program that can extract a table, view, or any custom query that contains a geometry column into a shapefile.*

USAGE:

```
pgsql2shp [<options>] <database> [<schema>.]<table>
pgsql2shp [<options>] <database> <query>
```

Basic Queries

How many points were used to digitize the road named “Cartmell Rd”?

```
SELECT ST_NPoints(the_geom)
FROM bc_roads
WHERE name = 'Cartmell Rd';
```

```
28
```

Basic Queries

How many holes or interior rings does the habitat area with hab_id = 891 (“Marbled Murrelet”) have?

```
SELECT ST_NumInteriorRings(the_geom)
FROM bc_habitat_areas
WHERE hab_id = 891;
```

```
5
```

Basic Queries

What is the length in kilometers of all roads named ‘Main St’?

```
SELECT sum(ST_Length(the_geom)) / 1000
          AS total_length
FROM bc_roads
WHERE name = 'Main St';

44.8489926296202
```

Basic Queries

What is the total area of all municipalities in hectares?

```
SELECT sum(ST_Area(the_geom)) / 10000
          AS total_length
FROM bc_municipalities;

1186482.22780945
```

Basic Queries

What is the average road length (treat every entry in the bc_roads table as a separate road)?

```
SELECT avg(ST_Length(the_geom))
FROM bc_roads;

434.230644730819
```

Basic Queries

How many roads are completely within the municipality of 'Hope‘?

```
SELECT count(*)
FROM bc_roads r,
     bc_municipalities m
WHERE ST_Within(r.the_geom, m.the_geom)
AND m.name = 'Hope';
```

```
280
```

Basic Queries

What is the latitude of the most southerly hospital in the province?

```
SELECT min(ST_Y(
          ST_Transform(the_geom, 4269)
       )) AS lat
FROM bc_hospitals;

48.4657953714625
```

Basic Queries

According to the datasets, is there a hospital in “Sicamous”?

```
SELECT count(*)
FROM bc_municipalities a,
     bc_hospitals b
WHERE a.name = 'Sicamous'
AND ST_Contains(a.the_geom, b.the_geom);

0
```

Basic Queries

What is the name of the closest hospital from Sicamous and how far away is it?

```
SELECT a.name, b.name,
       ST_Distance(a.the_geom, b.the_geom)
FROM bc_municipalities a,
     bc_hospitals b
WHERE a.name = 'Sicamous'
ORDER BY ST_Distance ASC
LIMIT 1;
```

```
  name    |              name                  | st_distance
----------+------------------------------------+------------
Sicamous | Shuswap Lake General Hospital | 22943.88318
```

Basic Queries

What are the names of the pubs located within 100 meters from "Granville St", located in downtown Vancouver?

```
SELECT DISTINCT c.name
FROM bc_municipalities a,
     bc_roads b,
     bc_pubs c
WHERE a.name = 'Vancouver'
AND b.name = 'Granville St'
AND ST_Contains(a.the_geom, b.the_geom)
AND ST_DWithin(b.the_geom, c.the_geom, 100)
ORDER BY c.name;
```

RODBD + RGDAL Libraries

- An ODBC library for the R environment.
- http://cran.r-project.org/web/packages/RODBC/
- RGDAL, an R Geospatial Data Abstracton Layer
- http://cran.r-project.org/web/packages/rgdal/

```
library(RODBC)
db <- odbcConnect('gisdatabase', uid='username', pwd='password')
sql <- 'select GeometryType(the_geom), NumGeometries(the_geom),
        asewkt(the_geom) as asewkt, gid from smr150_region'
res <- sqlQuery(db, sql)

geomtype <- as.character(res$geometrytype)
geomnum <- as.character(res$numgeometries)
geom <- as.character(res$asewkt)
geomdesc <- as.character(res$gid)

res <- sqlQuery(db, "SELECT attr, ST_X(the_geom) AS x,
                     ST_Y(the_geom) AS y FROM yourtable WHERE ...")
coordinates(data_frame) <- ~x+y

plot(coordinates)
```