

geodjango

“a world class geographic web framework”

django.contrib.gis

- geometry field
- (indexed) querying on geometries via the Django ORM
- multiple backends: PostGIS, MySQL, Oracle, spatialite
- importing (via GDAL) support
- distance API
- {google maps, open streetmap} view support
- geo IP location
- geo sitemaps
- admin support for all of this
- ctypes API for libGEOS and libGDAL
- format conversion

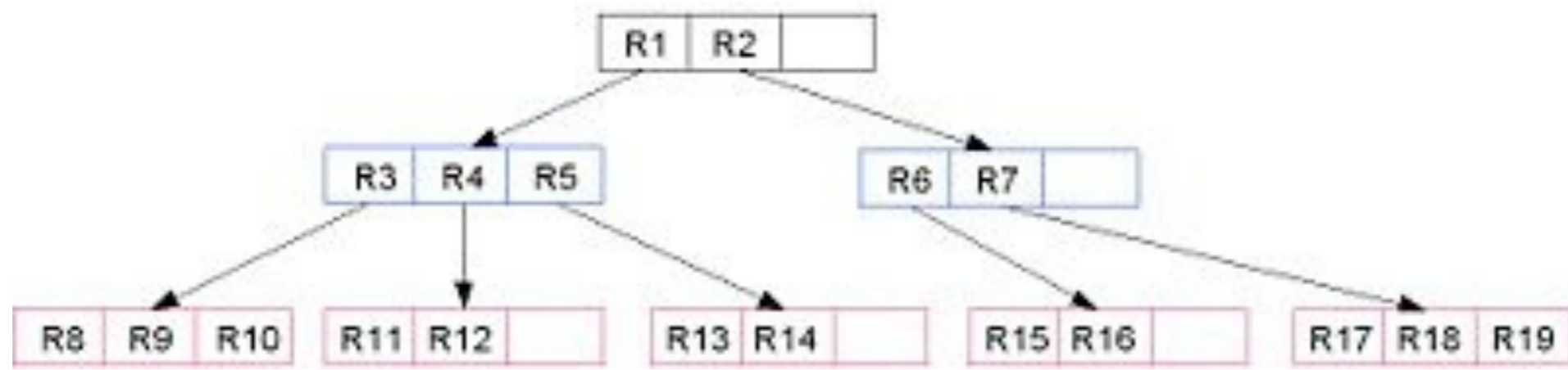
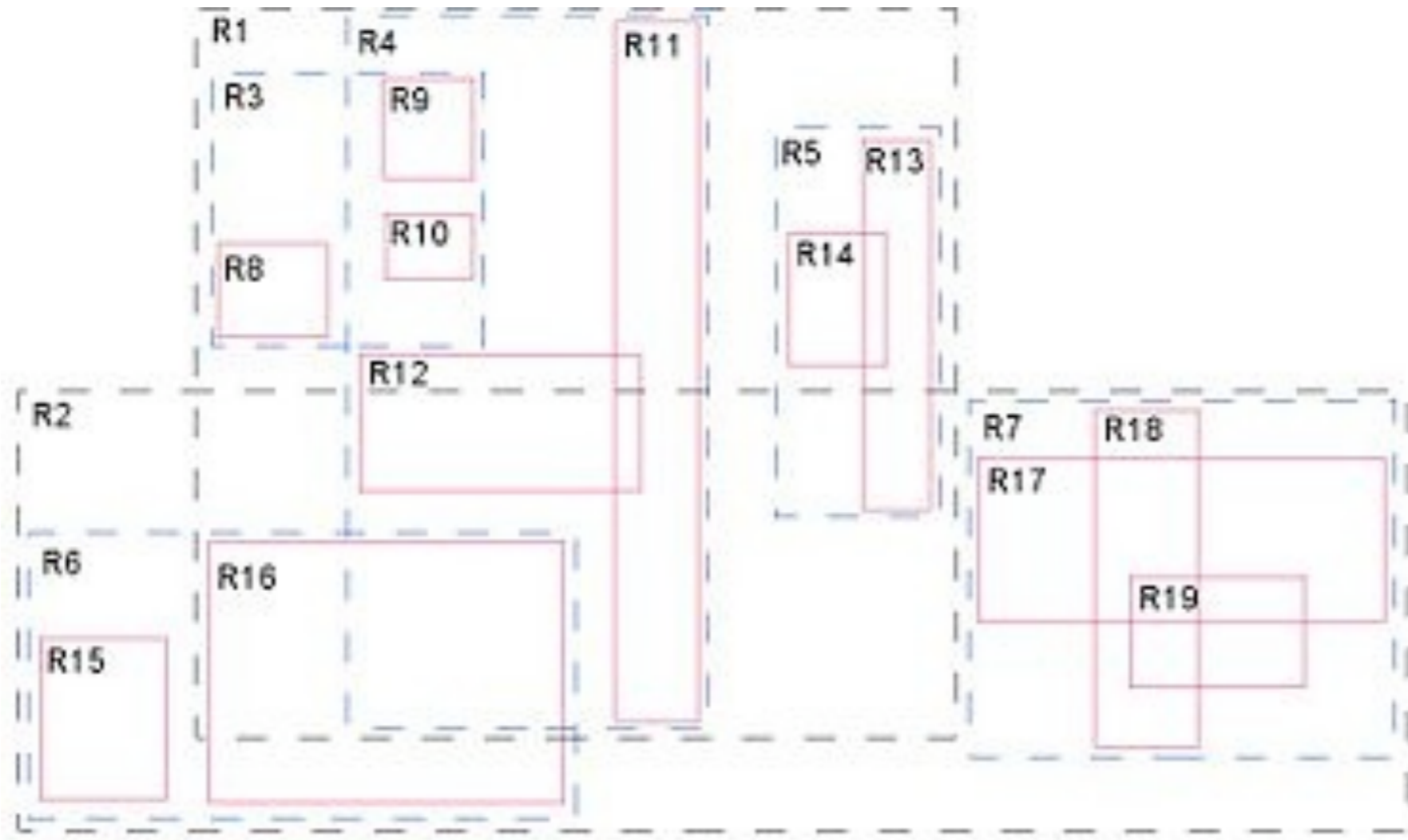
geospatial data

- any point on earth can be named by a (latitude, longitude)-pair.
- these are angles, not distances
- this can be significant: 0.02 degrees of longitude in San Francisco is 1,935m; in Nigeria it's 2,500m
- projected coordinate systems use distance units
- there are many types of projections,

```
django=> select count(*) from spatial_ref_sys;  
3162
```

- we'd like to store & query it: hierarchies, containment, distance, intersections, ...
- Postgres (PostGIS), MySQL, Oracle and SQLite (spatialite) all support it
- Though pretty much everybody uses PostGIS

- these add a field type, and a spatial index to their databases
- the index is an R-tree: stores minimum bounding rectangles (MBRs), organized so that each non-leaf node is the bounding box of all its children
- indexing of *points* can also be done by space-filling curves and a B-tree



data sources

- despite KML, geojson, etc., most data are still in somewhat arcane shapefile format, including most US government data
- GDAL is here to help: abstracts away data *sources*, presenting a uniform notion of layers
- geodjango helpfully includes GDAL ctypes wrappers

```
>>> from django.contrib.gis.gdal import DataSource  
>>> ds = DataSource('/tmp/data')
```

- can access layers, geometries, metadata
- *huge* number of supported formats
- geodjango takes it one step further: layer mapping generates models & imports data from any GDAL layer

data

- huge amount of public GIS data
 - census TIGER, USGS, etc.
 - municipalities/counties
 - flickr, zillow, etc.

geometries for your models

- geodjango provides full integration with the django ORM
- geometry fields, geometry manager, aggregates, etc.
- integrates fully with the query API, serialization, the works.

super easy*

```
from django.contrib.gis.db import models

class City(models.Model):
    geom = models.GeometryField()
    name = models.CharField(max_length=50)

    objects = models.GeoManager()

>>> python manage.py syncdb
```

*more or less: you may need to install PostGIS, setup template databases, etc.

manipulation

```
>>> from django.contrib.gis.geos import Point, Polygon,
MultiPolygon, GEOSGeometry as Geometry
>>> from django.contrib.gis.measure import D
>>>
>>> sf = City(
    name="San Francisco",
    geom=Polygon((( -123.173825, 37.63983),
                   (-123.173825, 37.929824),
                   (-122.28178, 37.929824),
                   (-122.28178, 37.63983),
                   (-123.173825, 37.63983))))
)
>>> sf.save()
```

```
>>> sf = City(
    name="San Francisco",
    # WKT
    geom=Geometry('POLYGON ((-123.17382499999999937
37.63983000000000035, -123.17382499999999937
37.92982400000000035, -122.28177999999999977
37.92982400000000035, -122.28177999999999977
37.63983000000000035, -123.17382499999999937
37.63983000000000035)))')
)
```

```
>>> sf = City(
    name="San Francisco",
    # GeoJSON
    geom=Geometry('{ "type": "Polygon", "coordinates":
[ [ [ -123.173825, 37.639830 ], [ -123.173825, 37.929824 ],
[ -122.281780, 37.929824 ], [ -122.281780, 37.639830 ],
[ -123.173825, 37.639830 ] ] ] }')
)
```

querying

```
>>> sf = City.objects.get(
    geom__contains=Point((-122.419204, 37.775196)))
>>> california = State.objects.get(name='California')
>>> cities = City.objects.filter(
    geom__within=california.geom
).distance(
    sf.centroid
)
>>> for city in cities:
    print city.name, city.distance # distance from SF
```

```
>>> Stores.objects.filter( # works on points only
    geom__distance_lt=(sf.centroid, D(km=100))
)
>>> City.objects.filter( # works on any geometry*
    geom__dwithin=(sf.centroid, D(km=1000))
)
```

*they have to be projected, so only really accurate for local projections, eg. say one for northern california

google maps in python

```
>>> from django.contrib.gis.maps.google import \
    GoogleMap, GPolygon
>>> polys = [GPolygon(city.geom, fill_color="#0f0f0f")
              for city in City.objects.all()]
>>> gmap = GoogleMap(polygons=polys)
```

in the template:

```
<head>
{{ gmap.xhtml }}
{{ gmap.style }}
{{ gmap.scripts }}
</head>

..
<div id={{ gmap.dom_id }}></div>
```

- zoom level determination based on data
- javascript via template, so you can customize your own

tying it all together: finding yuppies with the US census

- census data is aggregated by blocks, tracts, tract groups, “places”, ...
- grouped into *tables* of data
- cells contain population counts, of different tabulations
- census data is *extremely* detailed — many population statistics go down to blocks. 6 MB PDF to describe tables.

P41. AGE BY TYPES OF DISABILITY FOR THE CIVILIAN NONINSTITUTIONALIZED POPULATION 5 YEARS AND OVER WITH DISABILITIES [19]

Universe: Total disabilities tallied for the civilian noninstitutionalized population 5 years and over with disabilities

Total disabilities tallied:

Total disabilities tallied for people 5 to 15 years:

- Sensory disability
- Physical disability
- Mental disability
- Self-care disability

Total disabilities tallied for people 16 to 64 years:

- Sensory disability
- Physical disability
- Mental disability
- Self-care disability
- Go-outside-home disability
- Employment disability

Total disabilities tallied for people 65 years and over:

- Sensory disability
- Physical disability
- Mental disability
- Self-care disability
- Go-outside-home disability

P42. SEX BY AGE BY DISABILITY STATUS BY EMPLOYMENT STATUS FOR THE CIVILIAN NONINSTITUTIONALIZED POPULATION 5 YEARS AND OVER [49]

Universe: Civilian noninstitutionalized population 5 years and over

Total:

Male:

5 to 15 years:

- With a disability
- No disability

16 to 20 years:

- With a disability:
 - Employed
 - Not employed

No disability:

- Employed
- Not employed

21 to 64 years:

- With a disability:
 - Employed
 - Not employed

- each aggregation level (“summarylevel” in census speak), more or less, has a TIGER geometry
- TIGER is 2GB of shapefiles, with polygons for all of these
- great start for a *heatmap*

finding the yuppies

- for each tract, score a combination of census tables, normalized to overall tracts in the city
- use the TIGER geometries to display these on a map, graduated by their normalized scores
- plotted simply on a google map w/ client-side polygons
- nobody can agree on what a “city” is :-)