POSTGIS在互联网场景中的应用

冯若航 PostgreSQL DBA @ 探探

大纲

What: 互联网中的GIS需求

How: 怎样使用PostGIS解决需求？

Why: 为什么选择PostGIS？

互联网GIS应用开发轶事

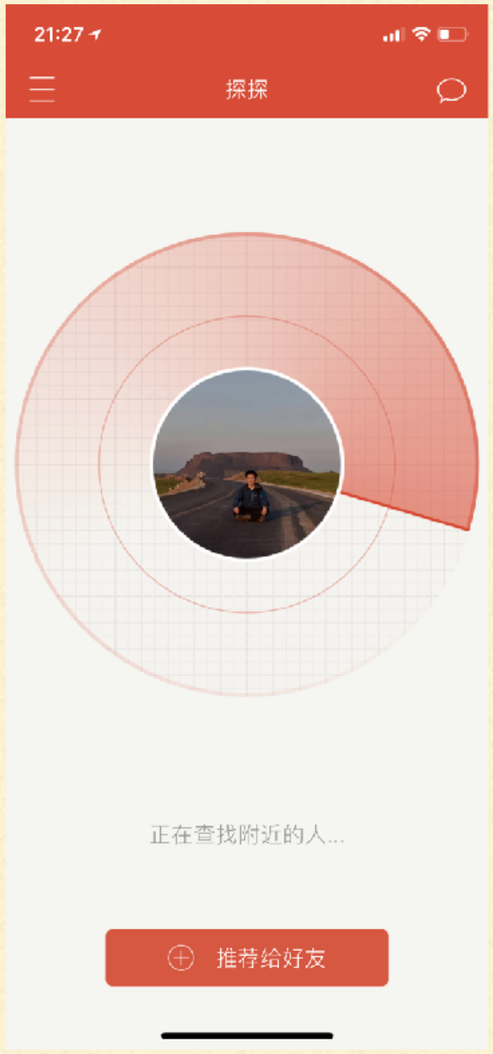
传统GIS


薛城工商GIS综合监管平台
物区分布
地图搜索


极端天气道路交通保畅物联网主题监测系统
降水监测
降雪监测
雾霾监测


My Data
Results
1 mile buffers - Police Stations
Geographies
States
Counties
ZIP Codes
Census Tracts
Census Block Groups

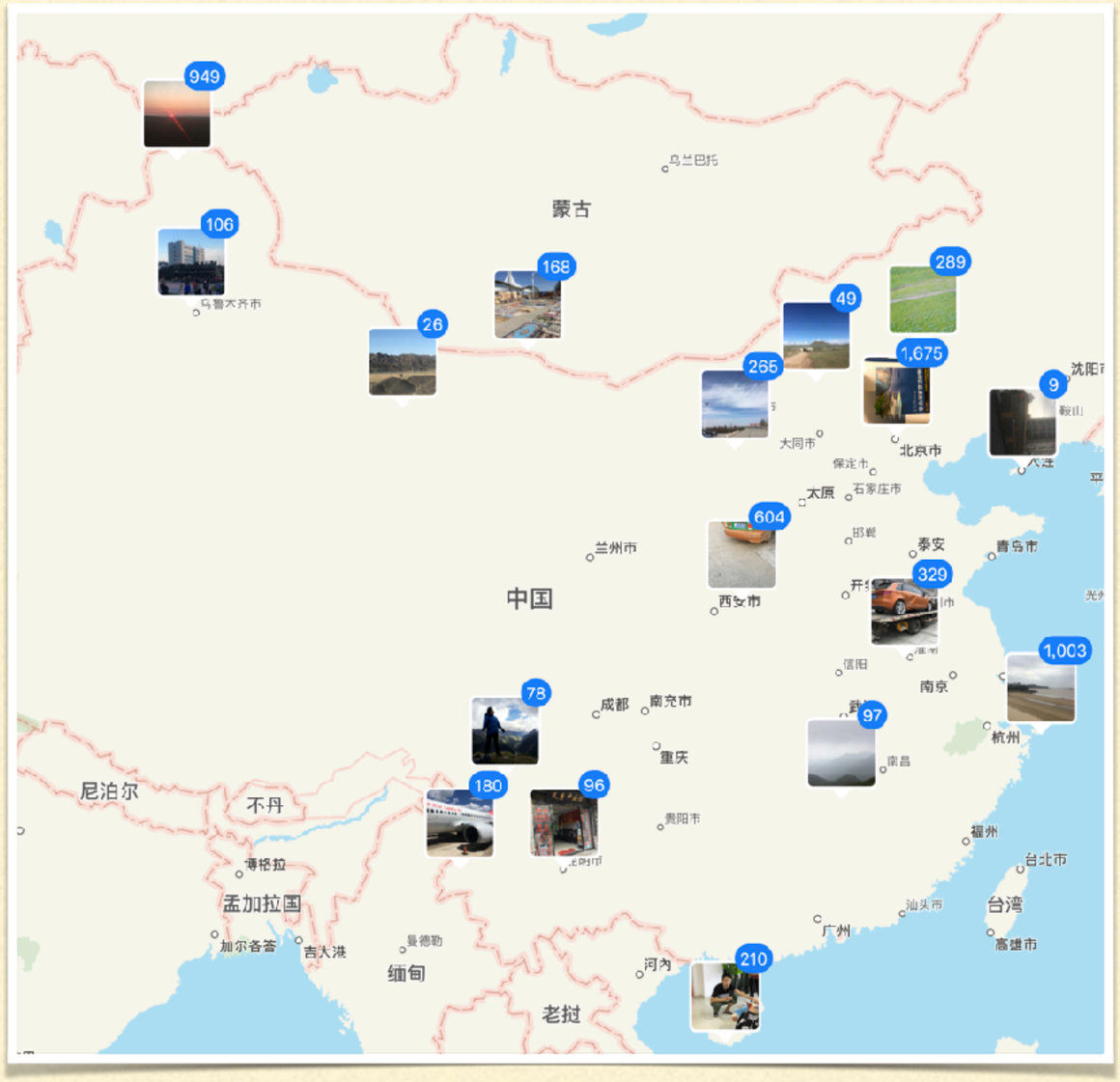
WHAT：互联网中的GIS需求

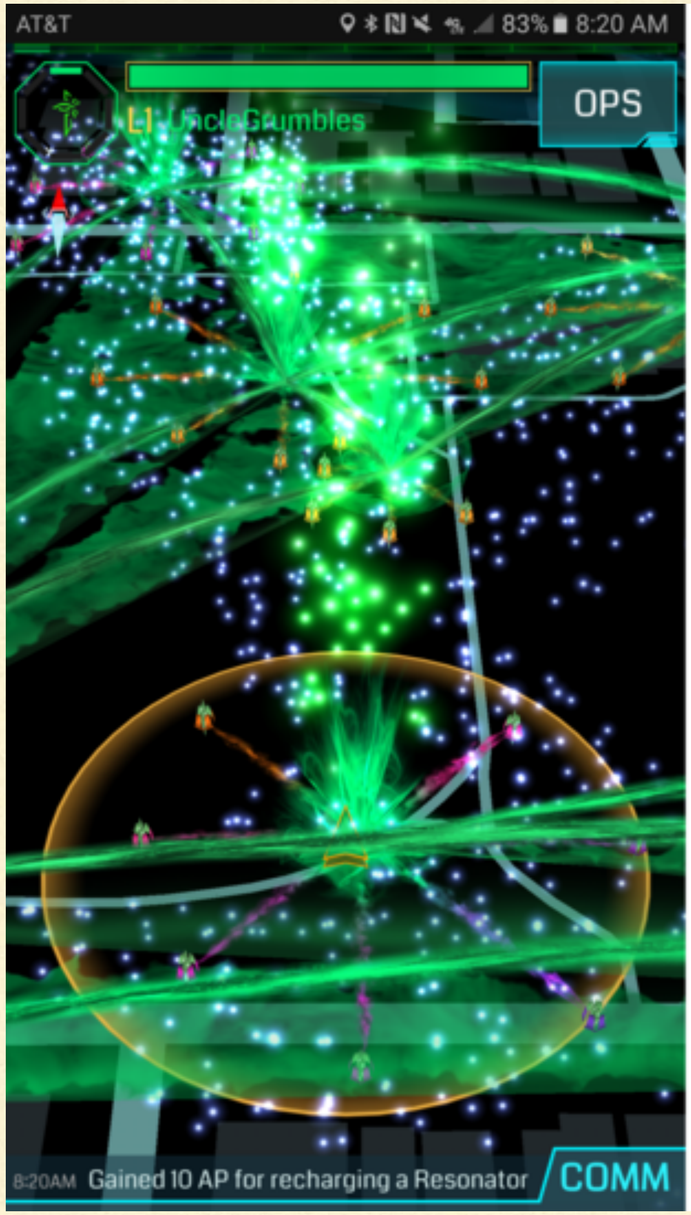


路线记录


7.3K/S
距离商家874m
瑞安市人民医院
院士路
薛前村
蔡宅村
11:15
预约送达
骑手正赶往商家 >
取消订单
催单
联系骑手
发红包
谁去拿外卖
玩游戏
投诉
一玩就嗨
一炮就爆
淘之米便当


无服务
19:37
59%
2 km
记录
按住统计设置按钮
0.1 公里/小时
目前的速度
0:00 分钟/公里
步伐


949
106
168
26
289
49
1,675
255
9
604
329
1,003
78
97
180
96
210
蒙古
中国
北京市
大同市
太原
石家庄市
兰州市
西安市
成都
南充市
重庆
南京
杭州
南昌
贵阳市
福州
台北市
台湾
高雄市
广州
汕头市
尼泊尔
不丹
孟加拉国
缅甸
老挝
河内


AT&T
83% 8:20 AM
OPS
L1 UncleGumbles
8:20AM Gained 10 AP for recharging a Resonator
COMM


AT&T
82% 8:21 AM
15
ahaywa

游戏

社交

(liao)

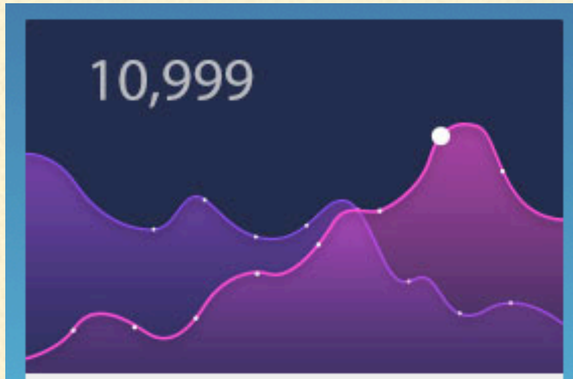
附近的人

同城交友

生人推荐

“2B”产品

客流画像分析——“用户大数据”

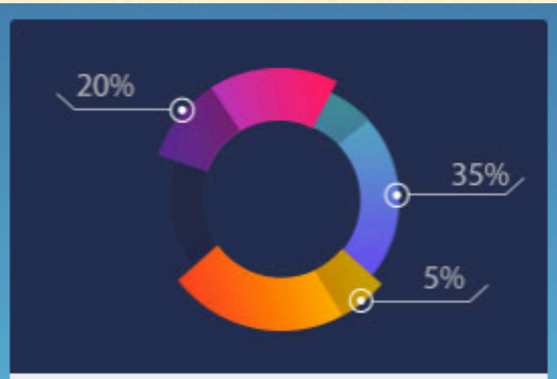


客流分析

清晰洞察区域以及周边客流趋势

区域内客流热力变化

多维度筛选目标客群流量



客群分析

全方位勾勒区域客群画像情况

区域客群到访分析

来源分析

客群居住地来源分析

客群工作地来源分析

商圈来源分析

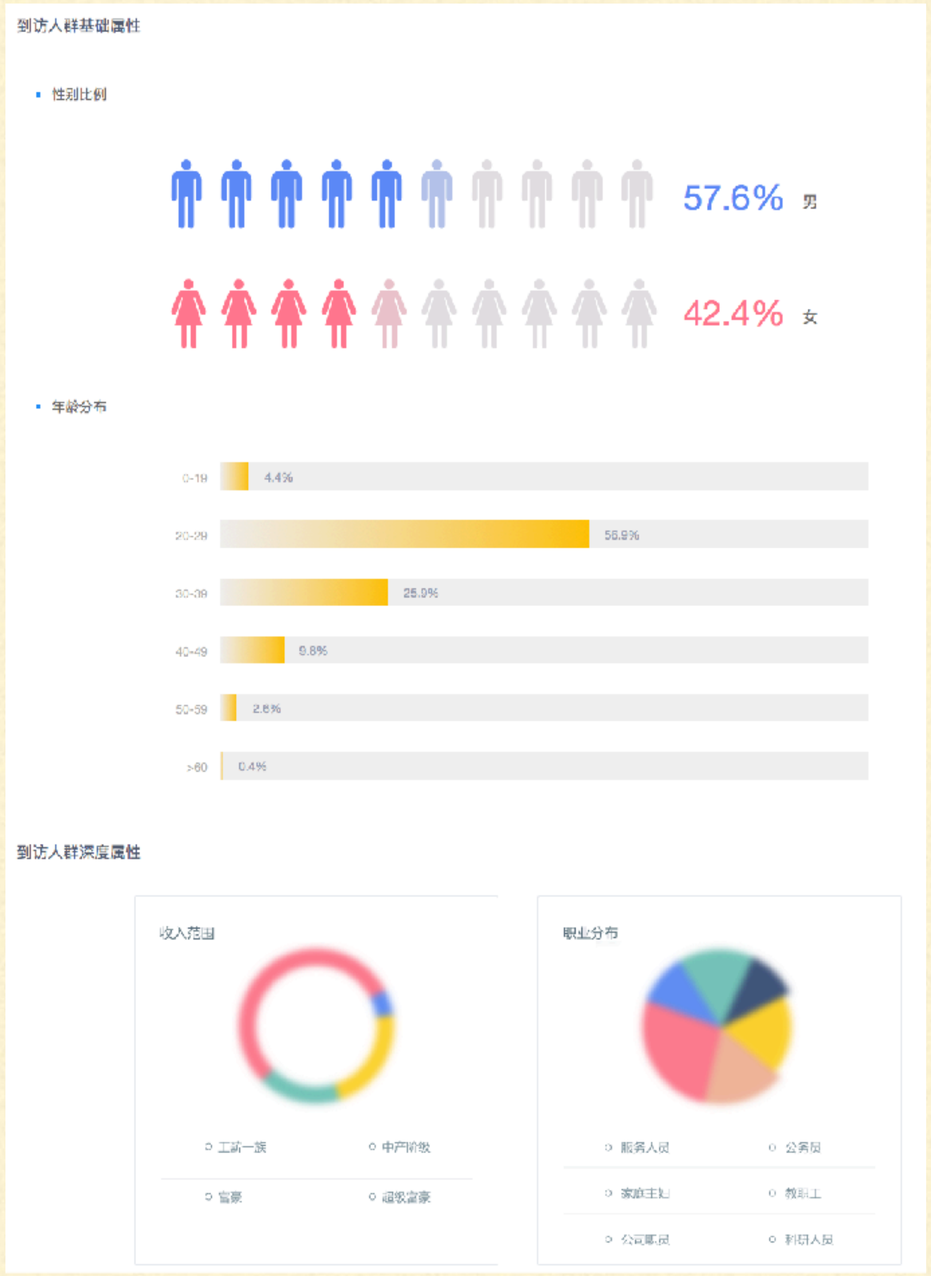
精准选址

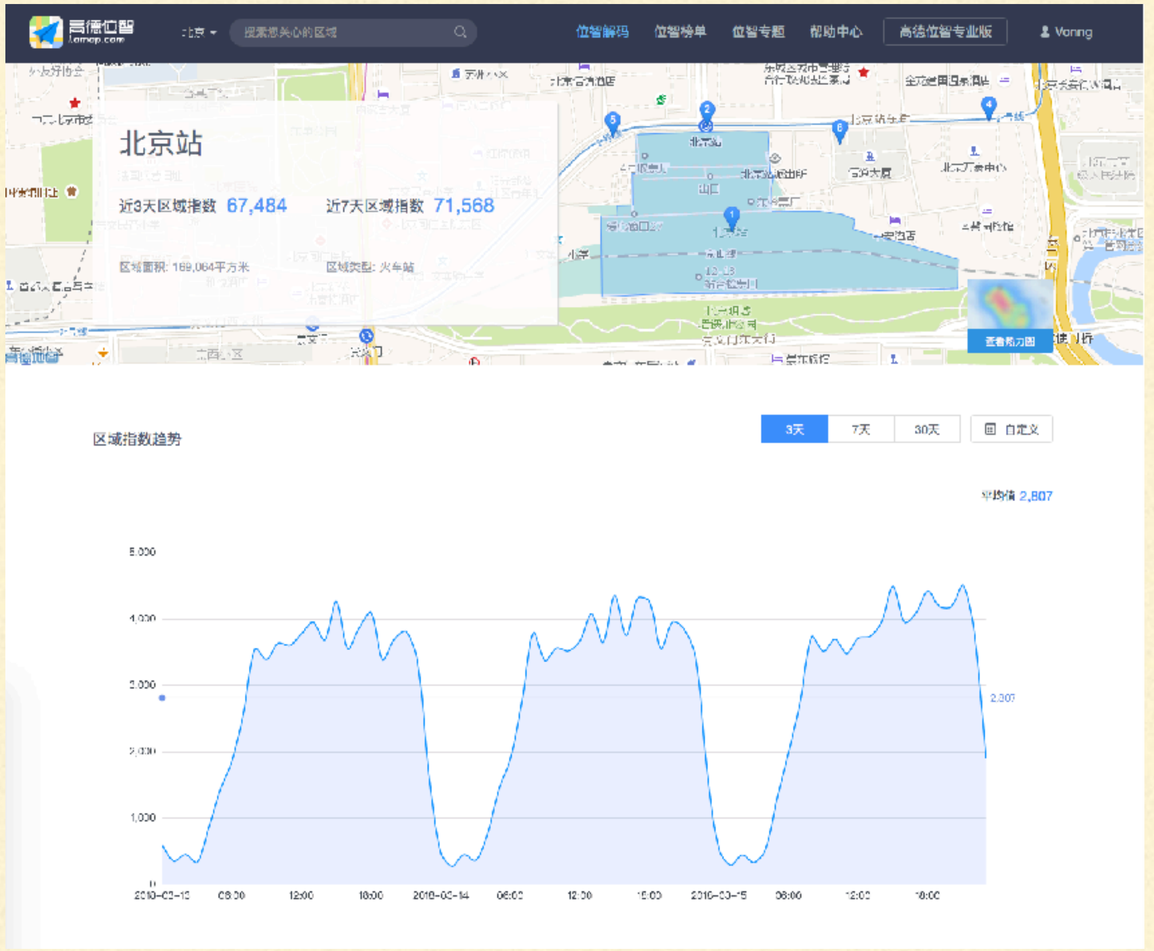
多维度画像筛选目标区域

竞争态势分析

全方位展示消费能力

北京站周边的客流与画像





这些应用的背后是什么?

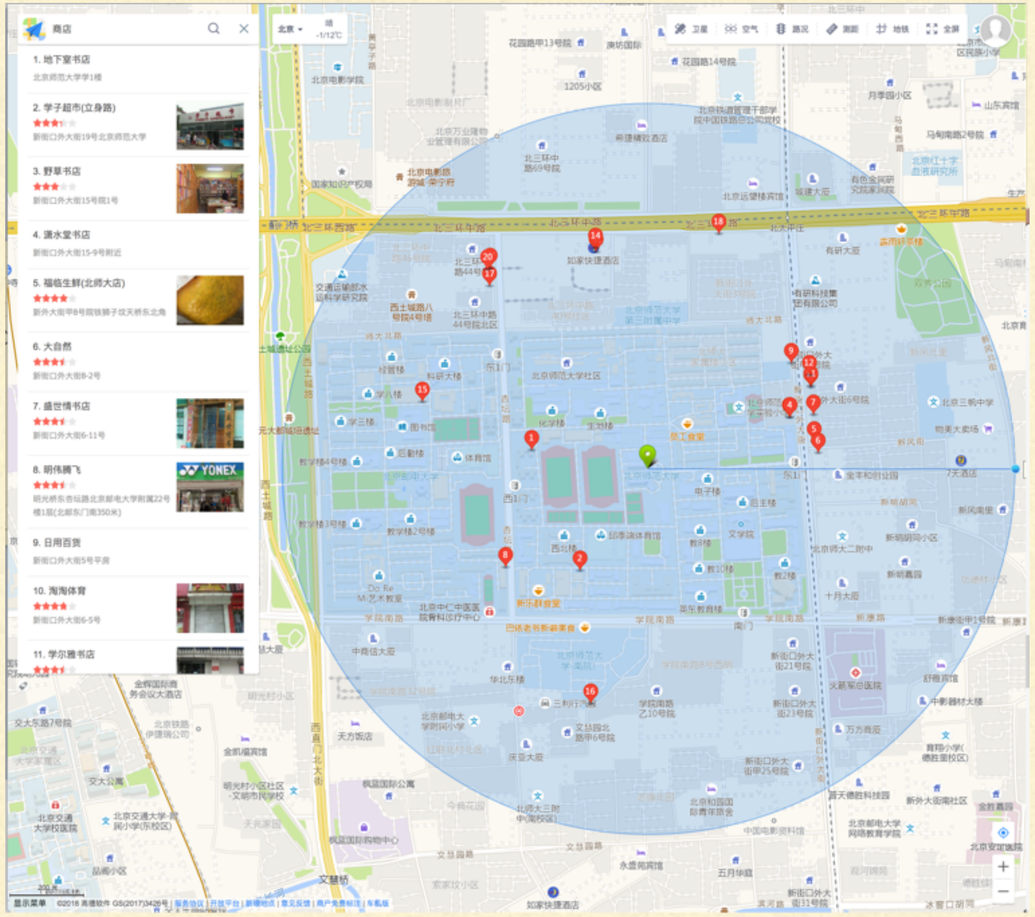
PostGIS

—— 相当一部分

定位、轨迹、导航、寻路

地理编码/逆编码、最近邻、地理围栏

How：怎样使用PostGIS解决实际需求？



觅食

谈朋友

典型场景： 最近邻搜索

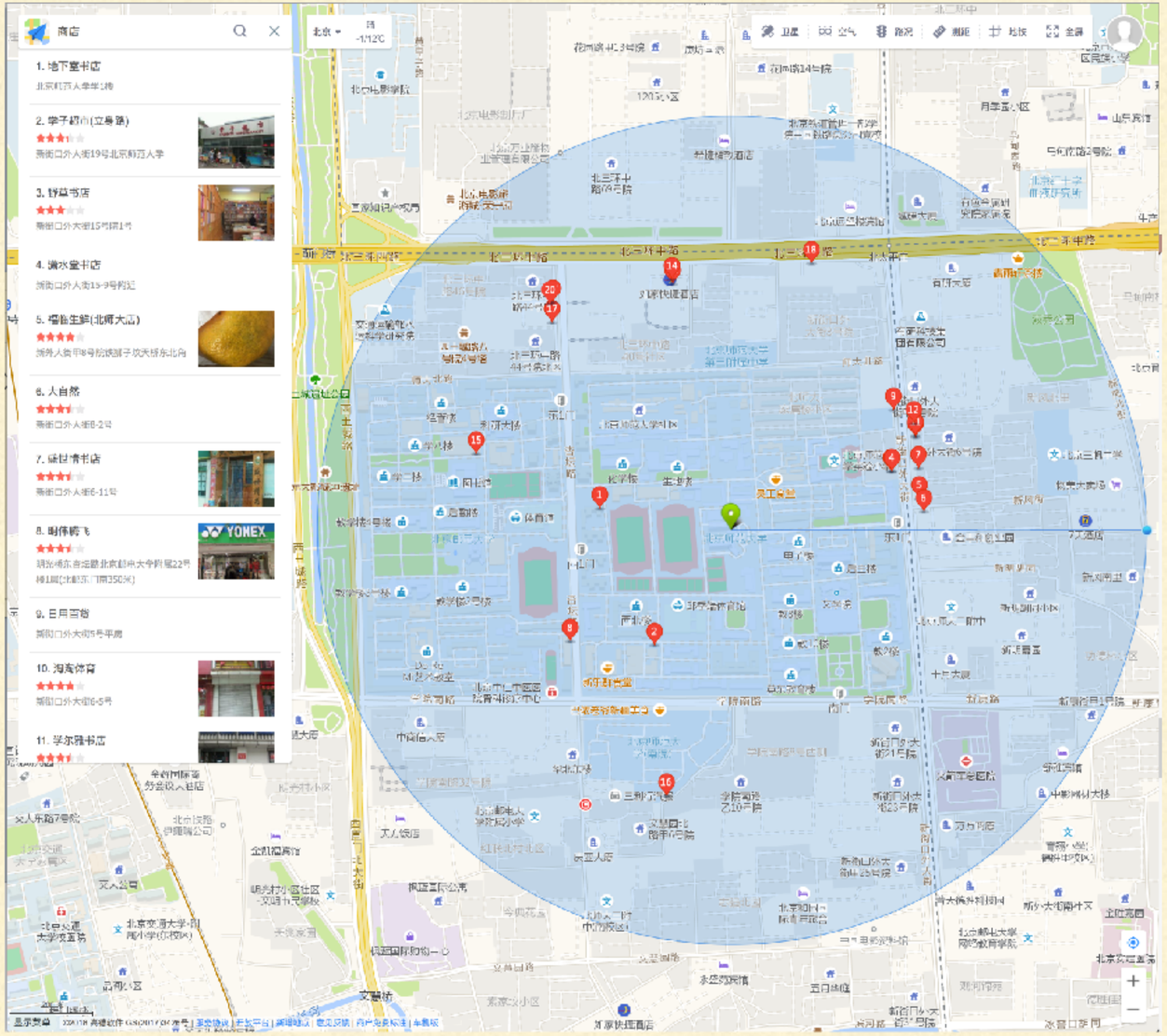
找到满足<某条件>，最近的K个对象

为用户推荐附近的POI（餐厅、公交站）

为用户推荐附近的用户（聊天匹配）

找到离用户最近的POI点（地理逆编码）

找到用户所在的省市区县（最近的面）



问题

给定中国所有POI点的表一张（一亿记录，含一千万餐馆）

给定中心点：北京师范大学 (116.3660, 39.9615)

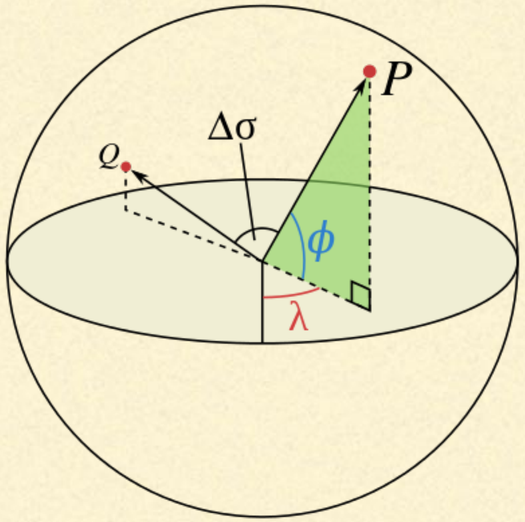
在**足够快**的时间内找出**距离**最近的10家餐馆，返回其名称和距离


```
CREATE TABLE pois (
  id          CHAR(10) PRIMARY KEY,
  name        VARCHAR(100),
  position    GEOMETRY, -- PostGIS ST_Point
  longitude FLOAT,      -- Float64
  latitude  FLOAT,      -- Float64
  category  INTEGER     -- type of POI
);
```

餐馆对应特征为： `WHERE category BETWEEN 50000 AND 51000`

距离如何定义？

欧式距离，球面距离，寻路距离，哈密顿距离.....



球面距离，单位为米

```
CREATE OR REPLACE FUNCTION sphere_distance(lon_a FLOAT, lat_a FLOAT, lon_b FLOAT, lat_b FLOAT)
  RETURNS FLOAT AS $$
SELECT asin(
           sqrt(
               sin(0.5 * radians(lat_b - lat_a)) ^ 2 +
               sin(0.5 * radians(lon_b - lon_a)) ^ 2 * cos(radians(lat_a)) * cos(radians(lat_b))
           )
       ) * 127561999.961088 AS distance;
$$
LANGUAGE SQL IMMUTABLE COST 100;
```

地球非球，乃不规则椭球。为了省事，设其为球。

足够快又是多快?

1 ms

作为参考，传统磁盘一次寻道是十几二十毫秒，
打王者荣耀的时候呢，一般延迟都是几十毫秒。

LEVEL-1 暴力扫表

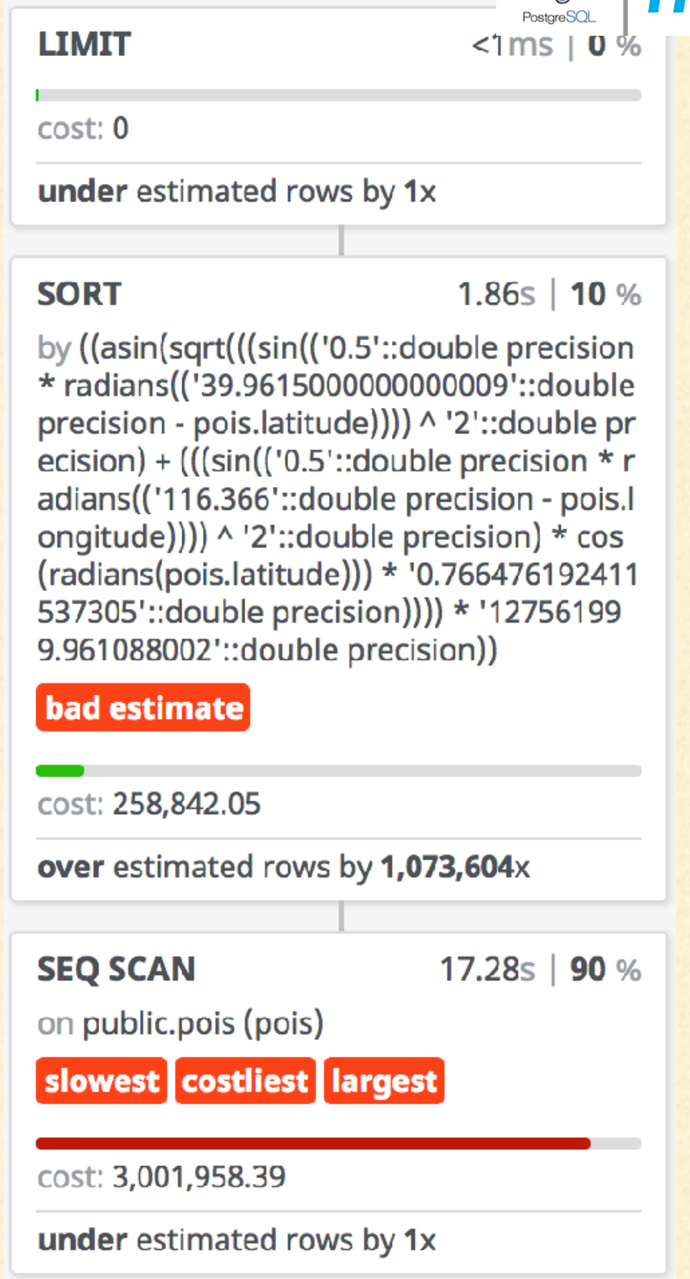
```
SELECT
   id,
   name,
   sphere_distance(longitude, latitude,
                   116.3660 , 39.9615 ) AS d
FROM pois
WHERE category BETWEEN 50000 AND 51000
ORDER BY d
LIMIT 10;
```

用时30秒

并行扫描17秒

黄花菜都凉了

name	distance
阿虾寿司	85.75879644790173
爱琴海咖啡	87.16563082548538
糊涂妹火锅	88.08752121954461
艺术休闲吧	89.78654594030888
兰婆婆湖南香辣米粉	91.93697623792761
千荷大煎饼	112.6598116413272
北师大北门湘菜馆	125.35558711224876
瑞合祥一品排骨	125.35558711224876
找你茶	148.79288130811864
玉林烤鸭	152.9514879867107

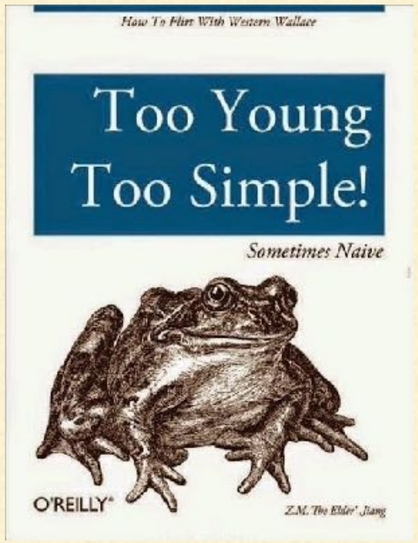


LEVEL-1 暴力扫表

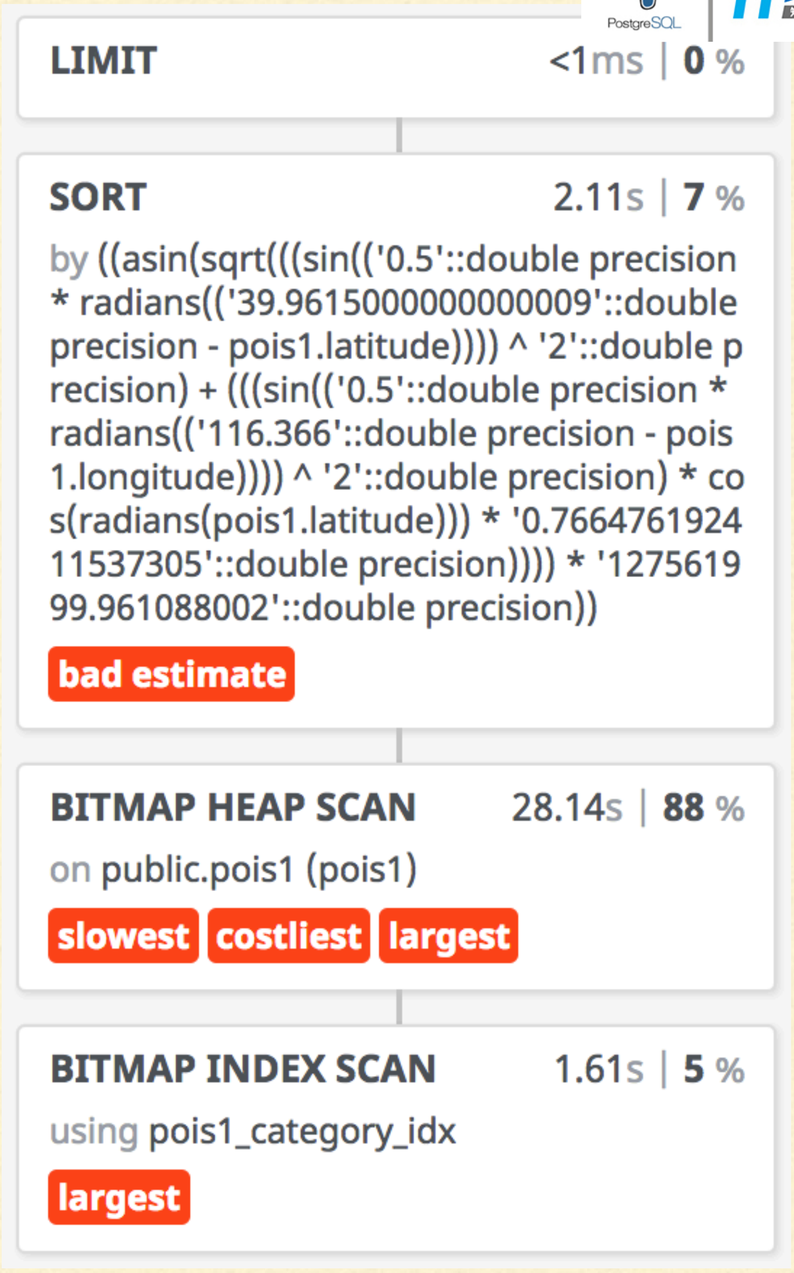
```
CREATE INDEX ON pois1 USING btree(category);
```

加个索引，顺序IO变随机IO

比顺序扫表还慢！



错误的索引还不如没有索引！



LEVEL-2 经纬索引

```
CREATE INDEX ON pois1 USING btree(longitude);
CREATE INDEX ON pois1 USING btree(latitude);
```

使用经纬度上的索引是基于这样一种思路：

如果我们用一个边长一公里的正方形（直径一公里的圆）去北师周围画个圈，别说十家餐厅了，一百家都有可能。

既然最近的十家餐厅一定落在这个框框内

那为什么不把这个框框外的点都先排除掉呢?

```
longitude BETWEEN  <left>    AND <right>  AND
latitude  BETWEEN  <bottom>  AND  <top>   AND
```

LEVEL-2 经纬索引

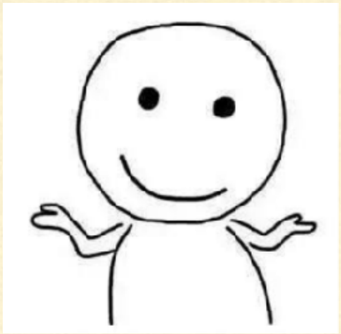
在暴力扫表的基础上，添加经纬度边界条件

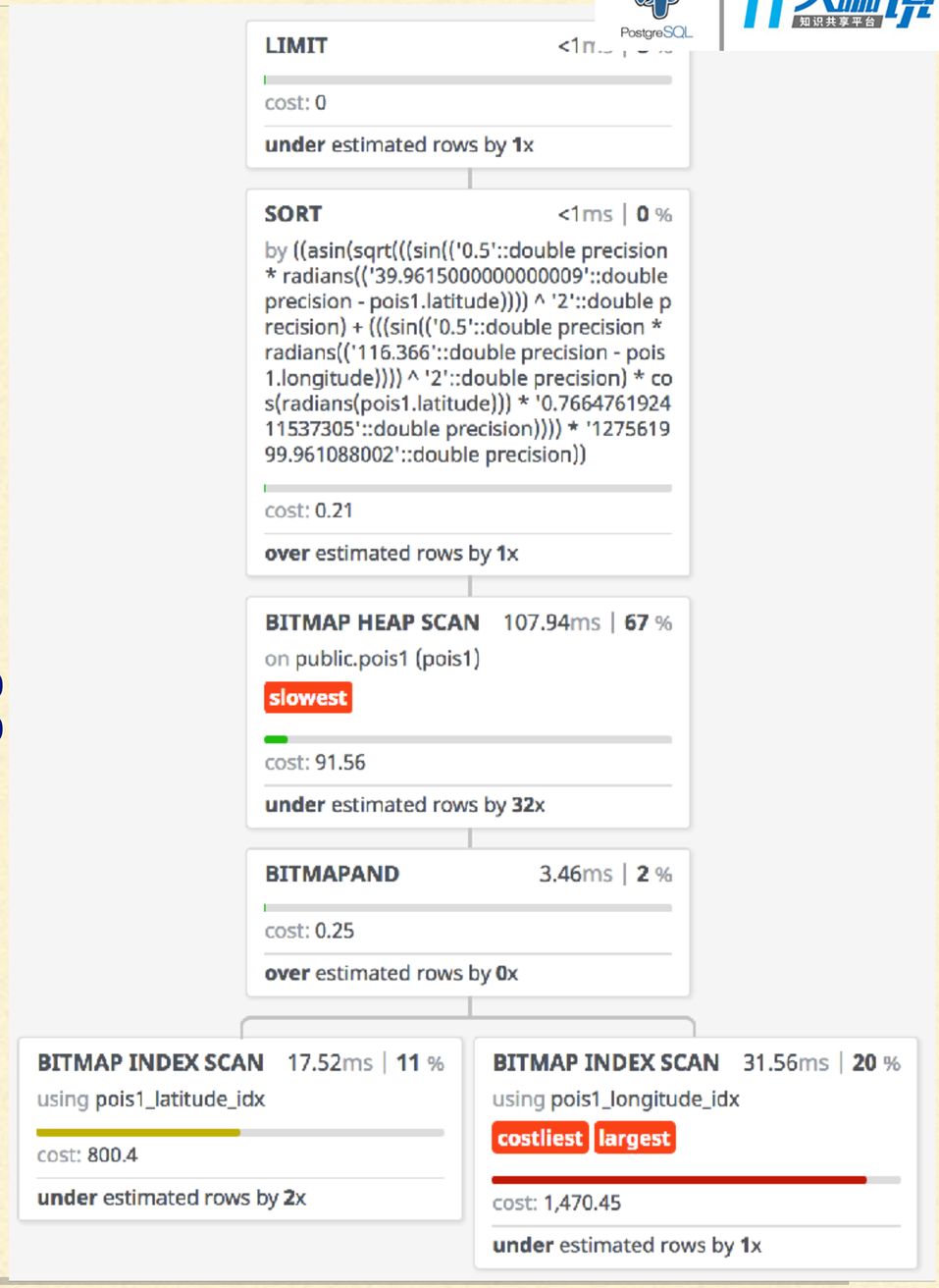
```
SELECT
  id, name,
  sphere_distance(longitude, latitude, 116.3660, 39.9615) as d
FROM pois1
WHERE
  longitude BETWEEN 116.3660 - 0.5 / 85  AND 116.3660 + 0.5 / 85  AND
  latitude  BETWEEN  39.9615 - 0.5 / 111 AND  39.9615 + 0.5 / 111 AND
  category  BETWEEN 50000                AND 51000
ORDER BY d LIMIT 10;
```

用时35毫秒

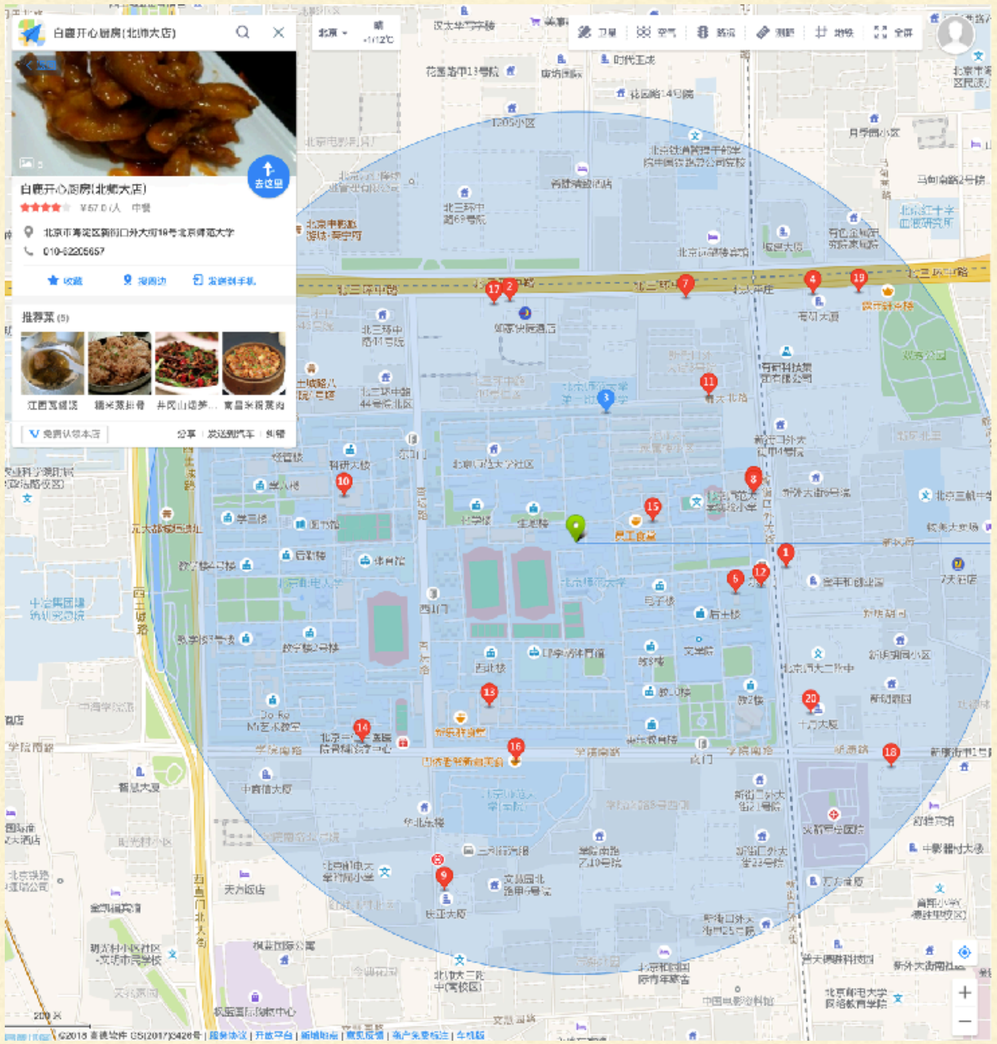
千倍提升，不错哦，但不能高兴的太早

这么多奇怪的常数又是几个意思?



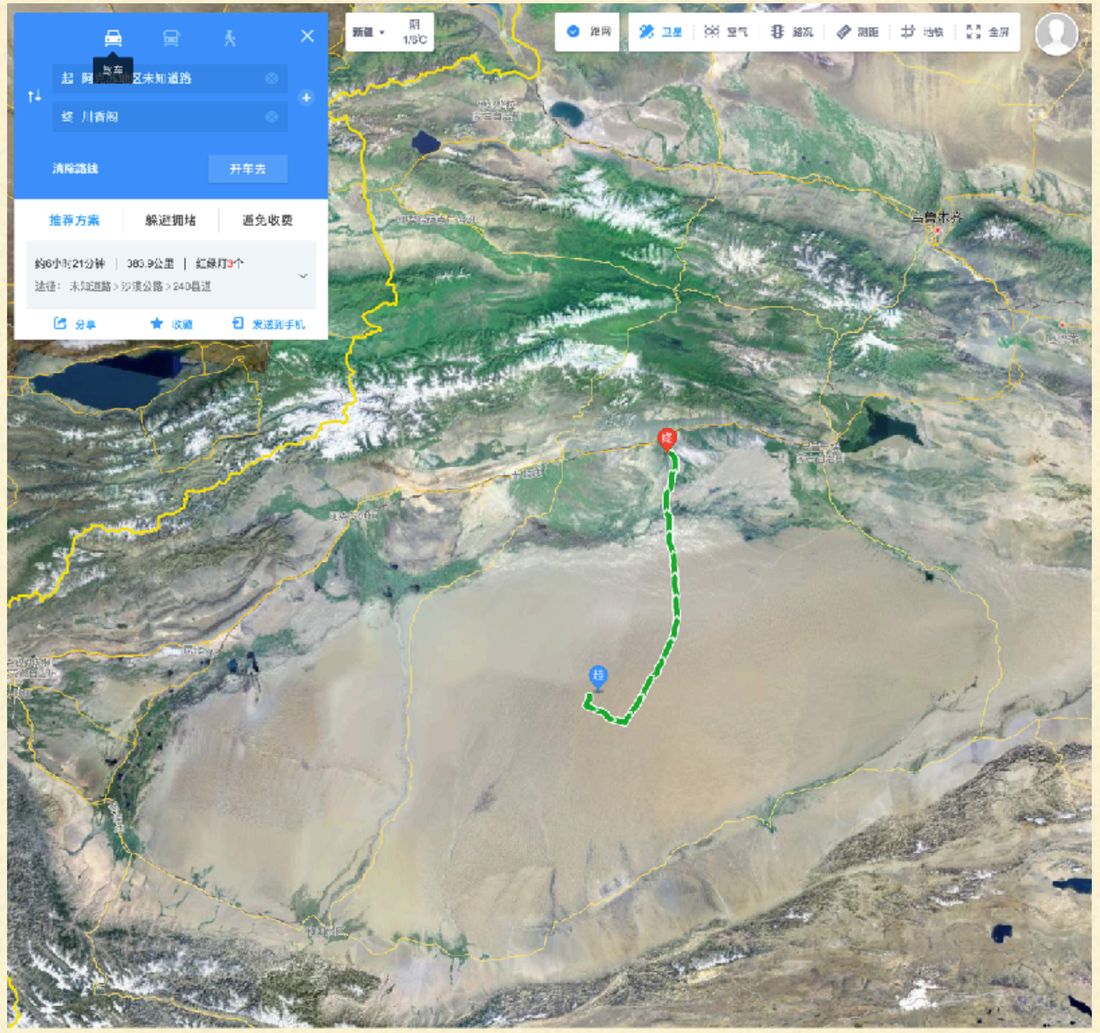


繁荣的五道口，一公里圈10家小意思。



半径大了性能差

300公里外才有一家，新疆人民买车在厕所



半径小了圈不着

LEVEL-3 多列索引与聚簇

```
CREATE INDEX ON pois4 USING btree(longitude, latitude, category);

CLUSTER pois4 USING pois4_longitude_latitude_category_idx;
```

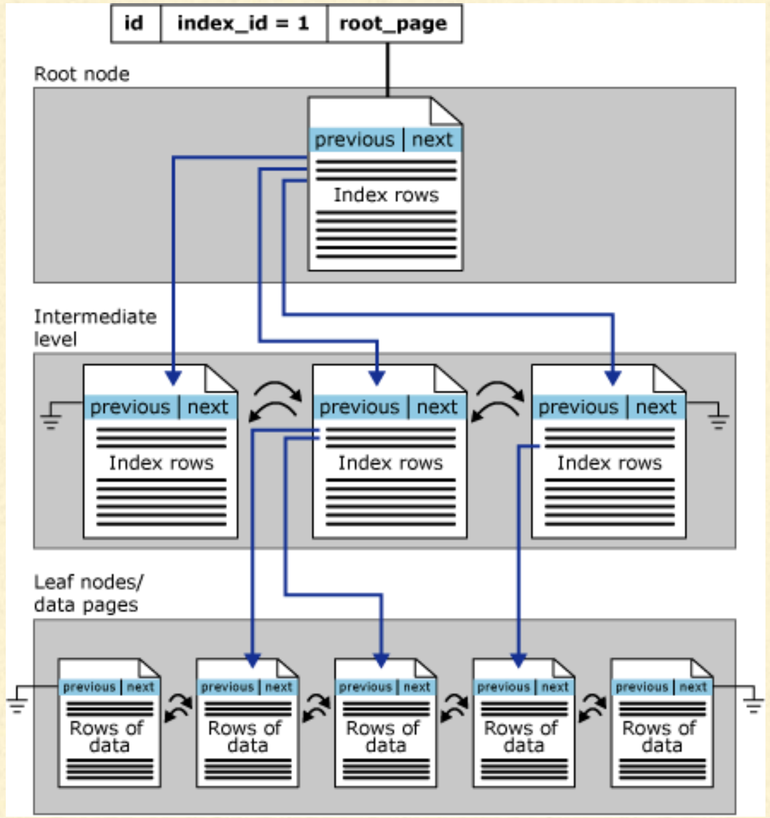
通常，一次查表只能用一个索引（并不是）

比起多个简单索引算做作图与或非

单个多列索引要快的多

让记录的物理存储顺序与这个索引保持一致

GeoHash与此类方法类似



LEVEL-3 多列索引与聚簇

仍然是一模一样的查询语句

从30毫秒提升到10毫秒，三倍的性能提升

对于传统关系型数据库，这差不多就是极限了

有没有优雅、正确、快速的解决方案呢?

LIMIT <1ms | **0** %

cost: 0

under estimated rows by **1**x

SORT <1ms | **1** %

by ((asin(sqrt(((sin(('0.5'::double precision * radians(('39.9615000000000009'::double precision - pois4.latitude)))) ^ '2'::double precision) + (((sin(('0.5'::double precision * radians(('116.366'::double precision - pois4.longitude)))) ^ '2'::double precision) * cos(radians(pois4.latitude))) * '0.766476192411537305'::double precision)))) * '127561999.961088002'::double precision))

cost: 0.49

over estimated rows by **2**x

INDEX SCAN 7.37ms | **98** %

on public.pois4 (pois4)
using pois4_longitude_latitude_category_idx

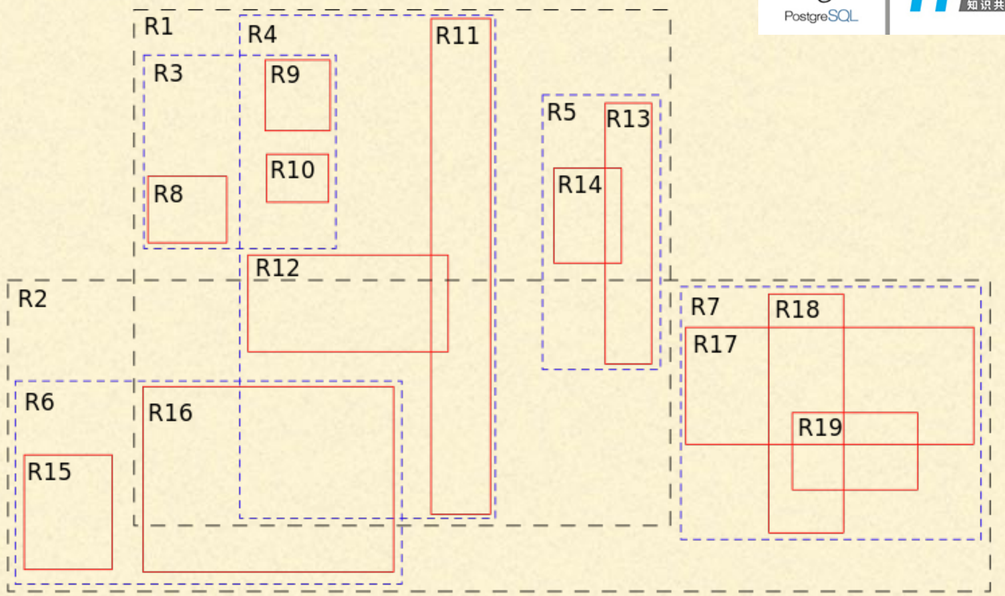
slowest **costliest** **largest**

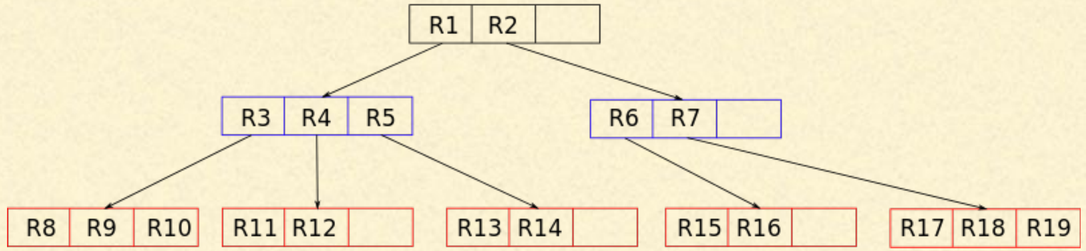
cost: 3,191.03

under estimated rows by **17**x

LEVEL-4 GIST

```
CREATE TABLE pois5
(
  id          CHAR(10),
  name        VARCHAR(100),
  position    GEOGRAPHY(Point, 4326),
  category    INTEGER
);
```






```
CREATE INDEX ON pois5 USING GIST(position);
```

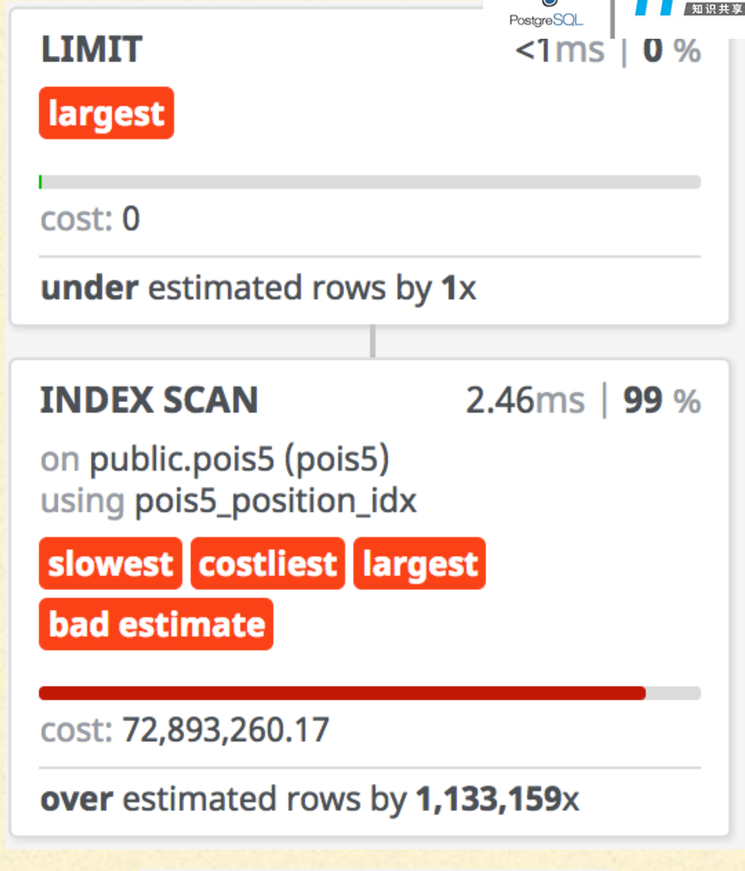
R树的核心思想是，聚合**距离相近**的节点，并在树结构的上一层，将其表示为这些节点的**最小外接矩形**，这个最小外接矩形就成为上一层的一个节点。因为所有节点都在它们的最小外接矩形中，所以跟某个矩形不相交的查询就一定跟这个矩形中的所有节点都不相交。

GiST 通用搜索树

LEVEL-4 GIST

```
SELECT
  id,
  name,
  position <->
  ST_Point(116.3660, 39.9615)::GEOGRAPHY AS d
FROM pois5
WHERE category BETWEEN 50000 AND 51000
ORDER BY d
LIMIT 10;
```

耗时3.4毫秒，比起传统方式提高了三倍性能

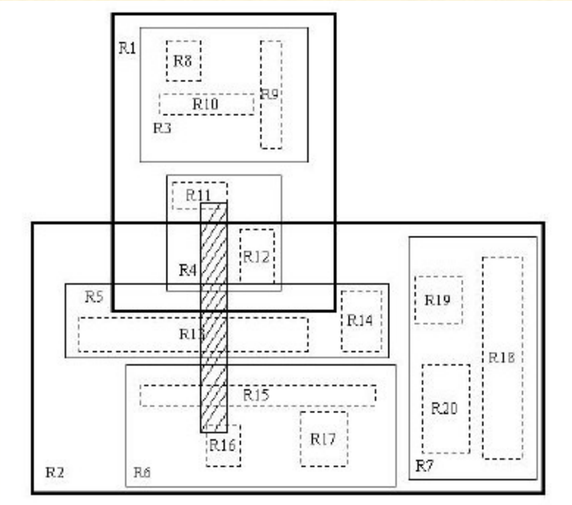


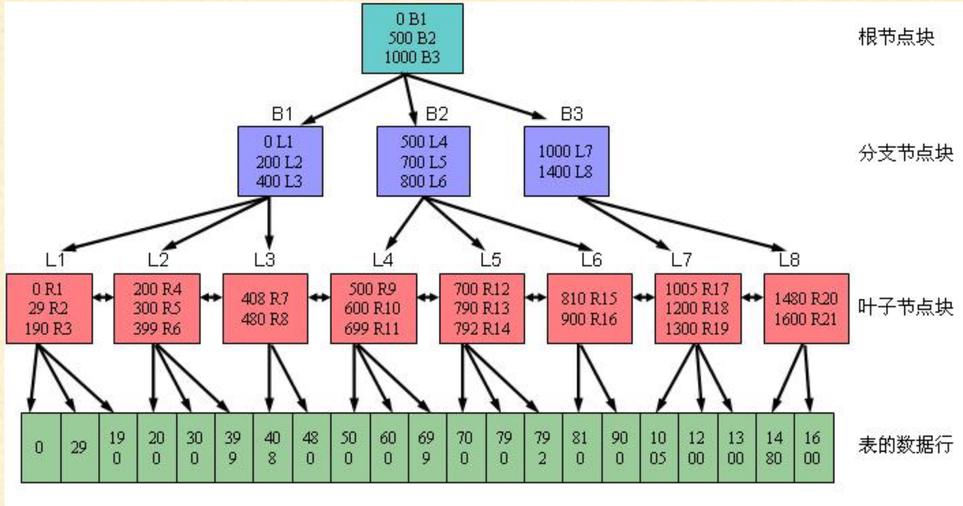
LEVEL-5 BTREE-GIST

观察Leve-4中的执行计划，我们发现category上的条件并没有用到索引

可不可以像Level-3中的优化方式一样，创建一个 position 与 category 的联合索引呢?

不幸的是，B树与R树是两种完全不同的数据结构，甚至连使用方式都不一样





用于空间类型的R树　　用于常规类型的B树

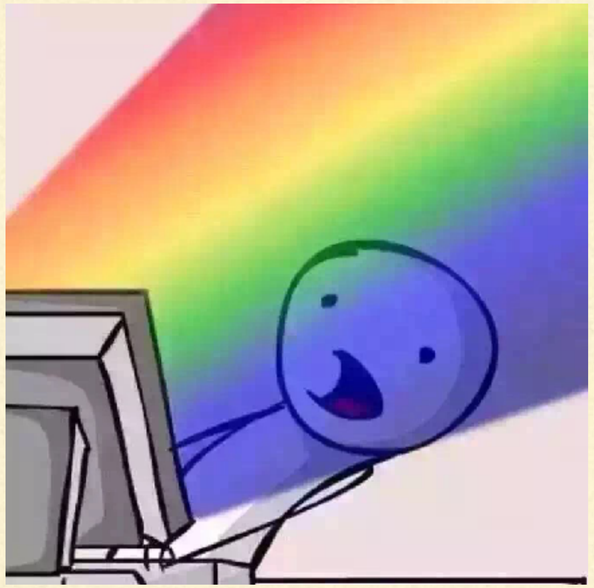
LEVEL-5 BTREE-GIST

PostgreSQL内置的扩展 btree_gist

允许创建常规类型与几何类型的联合索引

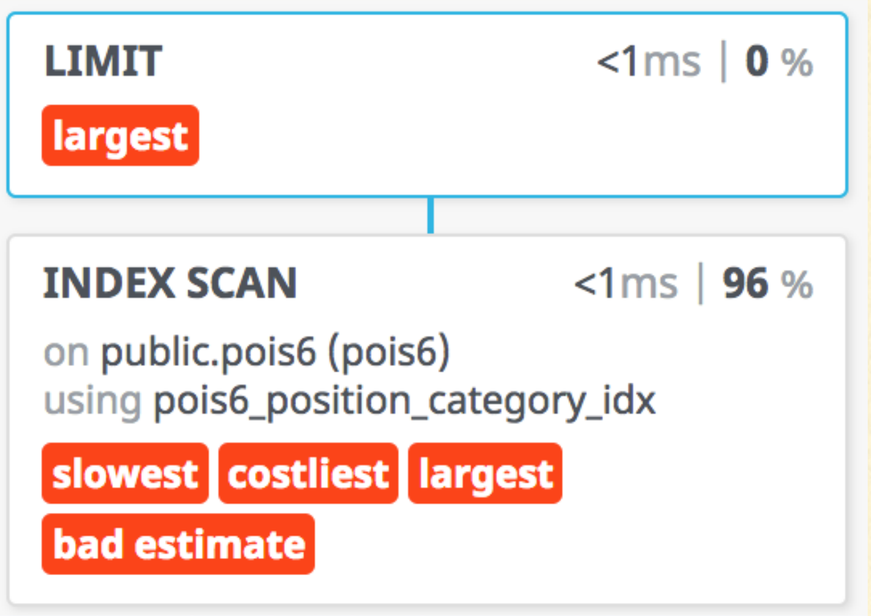
```
CREATE EXTENSION btree_gist;

CREATE INDEX ON pois6 USING GIST(position, category);
CLUSTER VERBOSE pois6 USING idx_pois6_position_category_gist;
```



1 毫秒

又是三倍性能提升



RECAP

Level	方法	性能/耗时(ms)	备注
1	暴力扫表	30,000	形式简单
2	经纬索引	35	额外复杂度
3	联合索引	10	额外复杂度
4	GIST	3.4	形式简单，距离更精确，PostgreSQL限定
5	`btree_gist`联合索引	1	形式简单，距离更精确，PostgreSQL限定

探探的负载量级与SLA

所有数据库查询峰值200万QPS

核心库32万QPS（1主10从）

单台机器峰值3万QPS，220台

关系型数据库总数据量 200 TB （以主库计）

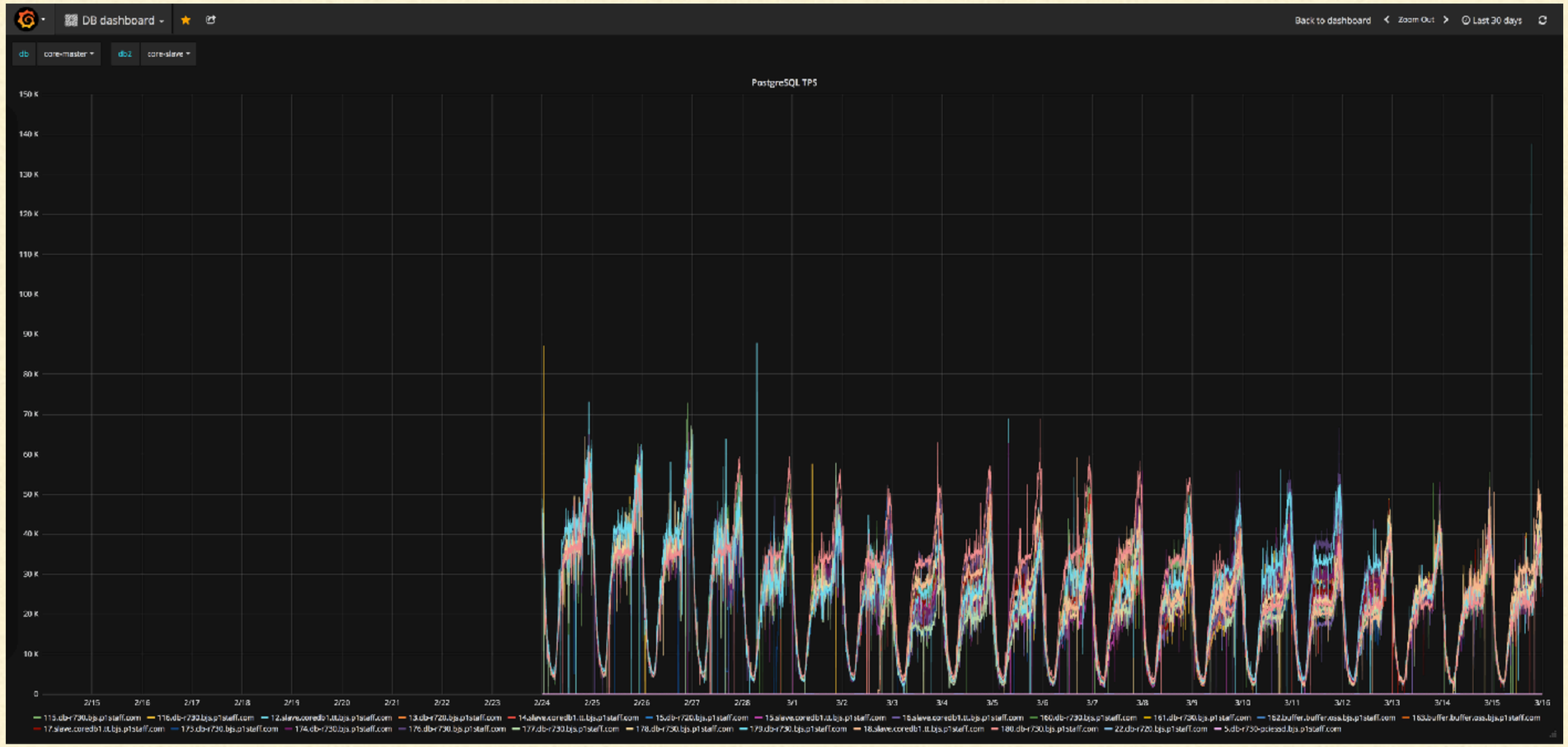
最多的函数调用均为PostGIS相关函数

SLA：99.99%的普通数据库请求要求在1毫秒内完成

220台数据库物理机，16组逻辑库

24核48线程 - 400G 内存

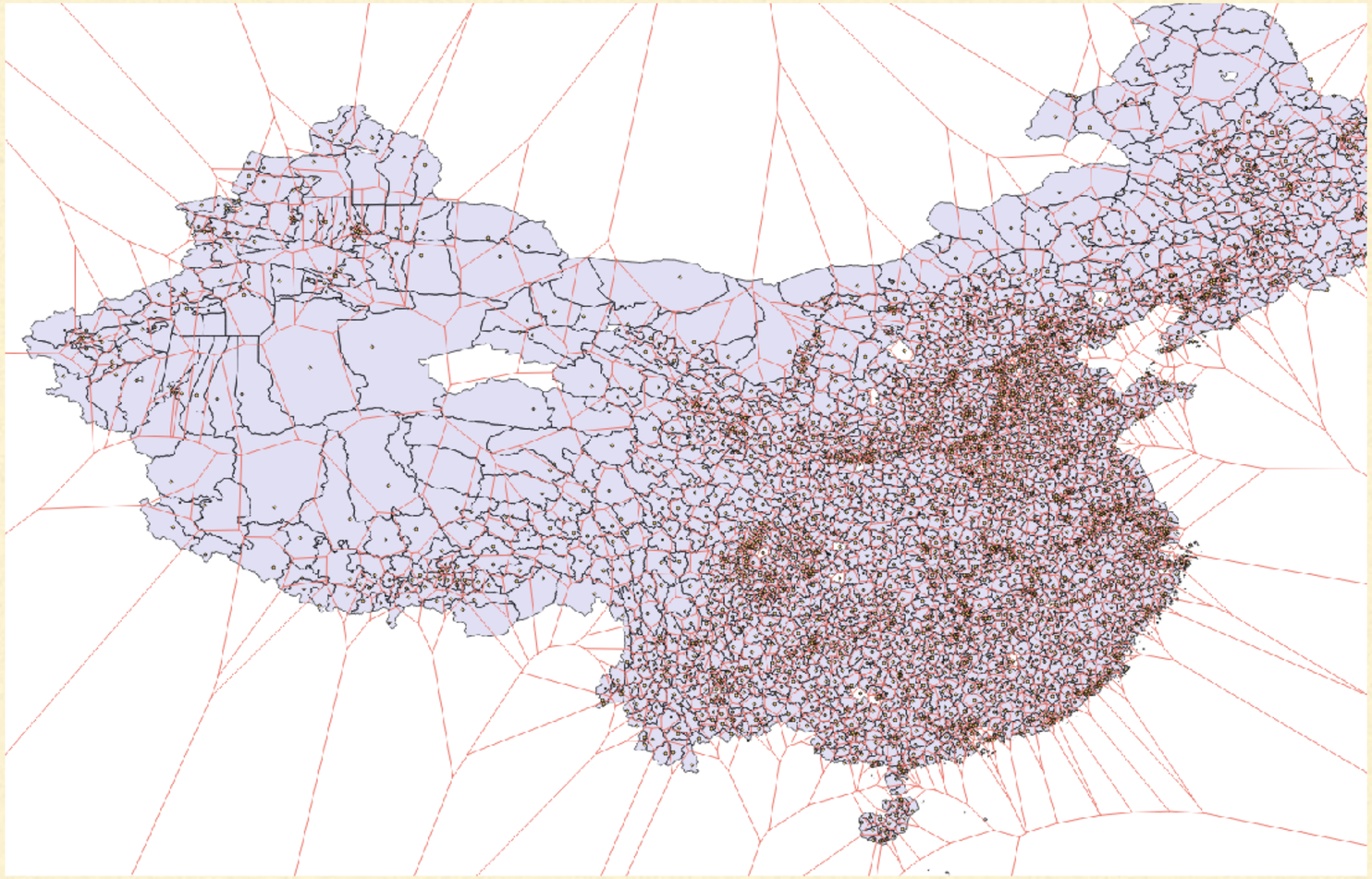
PCI-E SSD 3.2T



PG对于绝大多数应用整个生命周期的数据量，都是足够的。

更有挑战的查询

地理编码 —— 省市县精准定位



最近邻搜索—Vornoi 方法

速度很快，但不准确

有木有又快又好的方法?

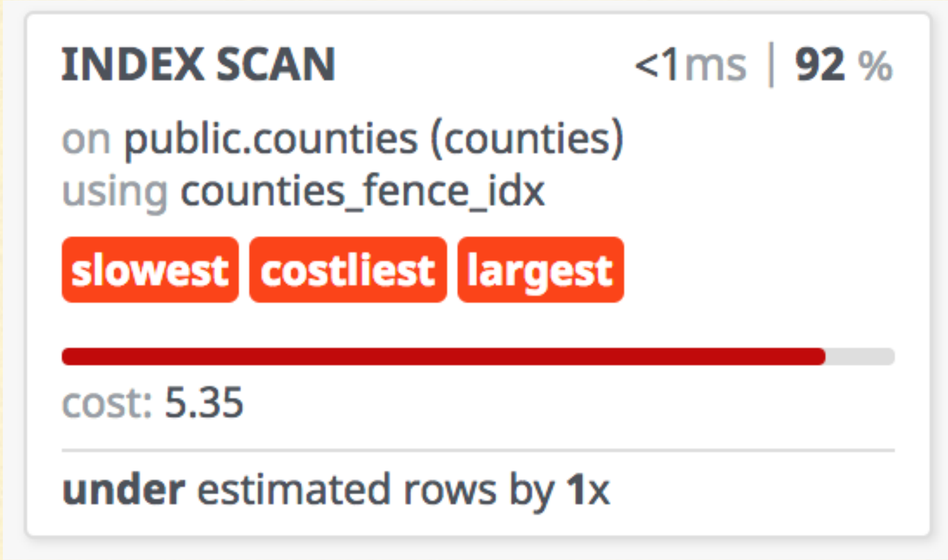

```
SELECT ST_VoronoiPolygons(ST_Union(center)) FROM counties;
```

采用包含判断

	adcode	province	city	county
1	533401	云南省	迪庆藏族自治	香格里拉市

```
CREATE TABLE counties
(
  adcode    INTEGER PRIMARY KEY,
  name      VARCHAR(64),
  province  VARCHAR(64),
  city      VARCHAR(64),
  county    VARCHAR(64),
  center    GEOMETRY,
  fence     GEOMETRY
);

CREATE INDEX ON counties USING GIST(fence);
```




```
SELECT adcode, province, city, county FROM counties
WHERE ST_WithIn(ST_Point(100, 28),fence);

SELECT adcode, province, city, county FROM counties
WHERE ST_Contains(fence, ST_Point(100, 28));
```

耗时0.6毫秒

完全达到生产水准

标题党：20行代码实现LBS应用后端

```
CREATE FUNCTION locate(lon FLOAT, lat FLOAT) RETURNS JSON
AS $$ SELECT row_to_json(l) FROM (
     SELECT adcode, province, city, county FROM counties
      WHERE ST_Contains(fence, ST_Point(lon, lat))
 ) l; $$ LANGUAGE SQL PARALLEL SAFE;
```

包成存储过程

4行SQL

```
import http, json, http.server, psycopg2

class GetHandler(http.server.BaseHTTPRequestHandler):
    conn = psycopg2.connect("postgres://localhost:5432/geo")

    def do_GET(self):
        args = [float(f) for f in self.path.lstrip('/').split('/')]
        self.send_response(http.HTTPStatus.OK)
        self.send_header('Content-type','application/json')
        with GetHandler.conn.cursor() as cursor:
            cursor.callproc('locate', args)
            result = print(json.dumps(cursor.fetchone()[0], ensure_ascii=False))
            self.wfile.write(result.encode('utf-8'))

with http.server.HTTPServer(("localhost", 3001), GetHandler) as httpd: httpd.serve_forever()
```

HTTP服务

15行Python

你甚至可以把这个脚本当成plpython3u的存储过程，放在PostgreSQL中执行，HTTP Server in Database！(Bad idea)

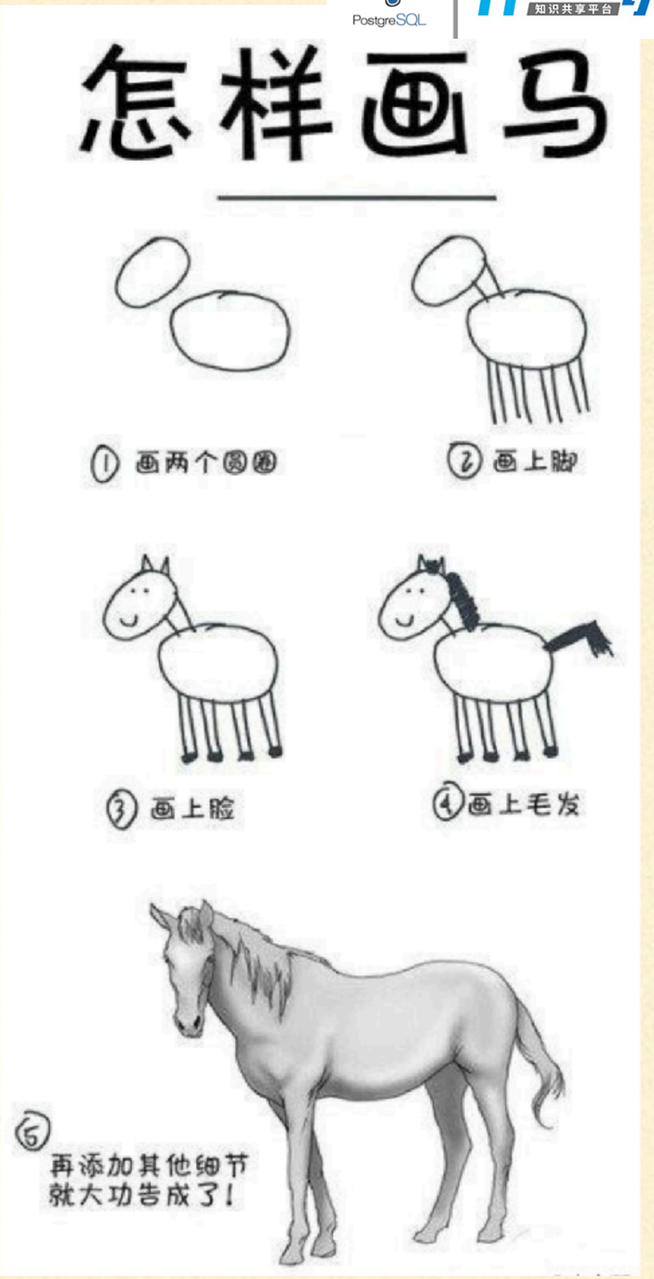
加上前端的各种处理、可视化等，大功告成。

```
$ curl localhost:3001/100/28
{"adcode": 533401, "province": "云南省", "city": "迪庆藏族自治州", "county": "香格里拉市"}

$ curl localhost:3001/100/24
{"adcode": 530902, "province": "云南省", "city": "临沧市", "county": "临翔区"}

$ curl localhost:3001/100/29
{“adcode": 513336, "province": "四川省", "city": "甘孜藏族自治州", "county": "乡城县"}

$ curl localhost:3001/111/22
{"adcode": 440981, "province": "广东省", "city": "茂名市", "county": "高州市"}
```



探探的PostgreSQL实践

重度使用存储过程： 80%以上的业务逻辑使用存储过程实现

SQL代码行数与后端Go代码行数基本相同

数据库的瓶颈已经由IO变为CPU

可维护性（成本）/ 可靠性（正确性）

PostGIS相对应用层解决方案的对比

最小表达力原则	一行SQL顶千行C++
千行代码缺陷率	代码越多，错误越多
管理复杂度	Keep it Simple，抽象
灵活性	快速做原型，拥抱变化

Why: 为什么选择PostGIS?

* 可扩展性（性能）
* 可维护性（成本）
* 可靠性（正确性）

扩展：互联网GIS应用开发杂谈　　一些经验

* 坐标系：火星坐标系二三事
* SDK　：地图功能哪家强?
* 可视化：如何包装工作成果
* 爬数据：巧妇难为无米之炊
* 薅数据：互联网的阴暗后厨
* G or IS？ PostGIS 的学习方法

火星坐标系二三事

我们常说的坐标系有哪些?

WGS84：为一种大地坐标系，也是目前广泛使用的GPS全球卫星定位系统使用的坐标系。
GCJ02：又称火星坐标系，是由中国国家测绘局制定的地理坐标系统，是由WGS84加密后得到的坐标系。
BD09：为百度坐标系，在GCJ02坐标系基础上再次加密。其中bd09ll表示百度经纬度坐标，bd09mc表示百度墨卡托米制坐标。

百度地图使用什么坐标体系?

使用百度地图的服务，需使用BD09坐标。
若使用非BD09坐标、未经过坐标转换（非BD09转成BD09）直接叠加在地图上，地图展示位置会偏移，因此通过其他坐标（WGS84、GCJ02）调用服务时，需先将其他坐标转换为BD09。

港澳台及海外，百度地图返回什么坐标?

中国地区(包括港澳台)，百度地图开放平台的所有产品，都支持返回GCJ02坐标系、BD09坐标系。
海外地区，目前返回的是WGS84坐标。

非百度坐标系，如何转换成百度坐标系?

我们全面支持GCJ02坐标系，并提供非百度坐标转换为百度坐标的方法。在使用服务时，如未对坐标参数进行设置（入参和出参，以各服务接口文档为准），默认使用BD09坐标系。

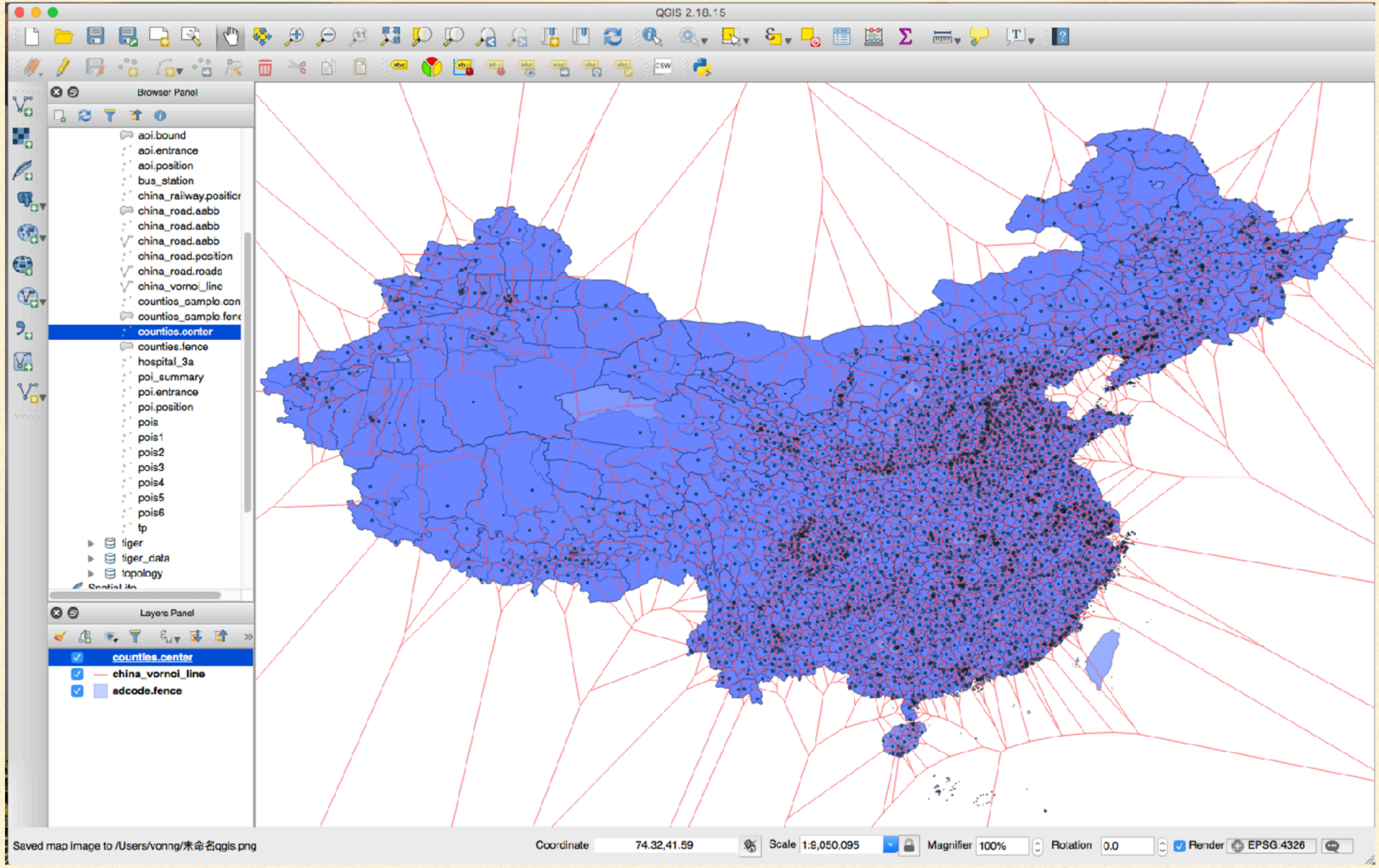
如果您想调用服务器端的坐标转换方法，请参考坐标转换API
如果您想在JS的前端网页中使用坐标转换功能，请参考JavaScript API坐标转换示例
如果您想在Android终端系统上使用坐标转换功能，请参考Android地图SDK坐标转换开发指南
如果您想在iOS的终端系统上使用坐标转换功能，请参考iOS地图SDK坐标转换开发指南

可视化

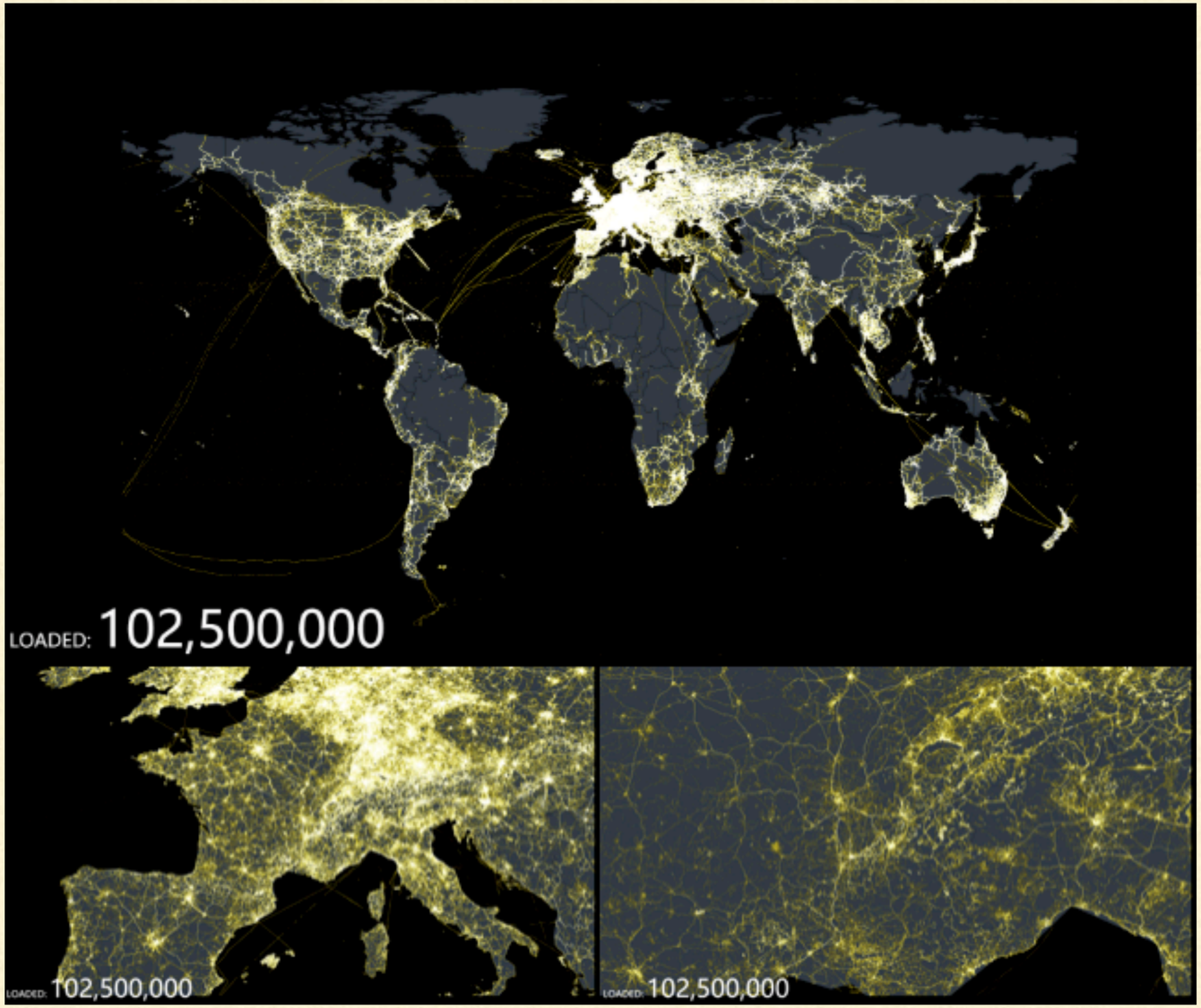
QGIS

慢、丑

不过还挺好用

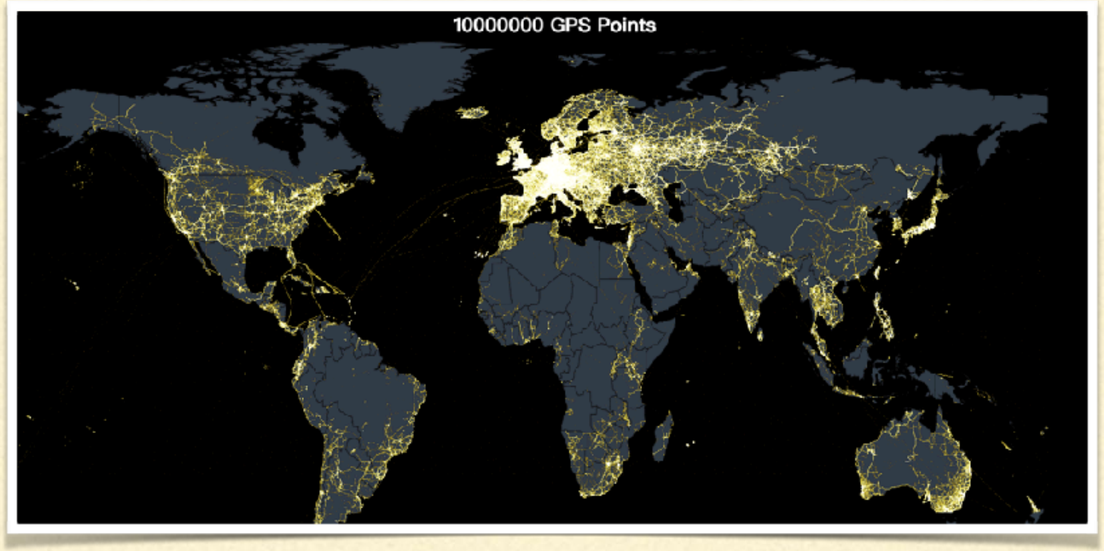


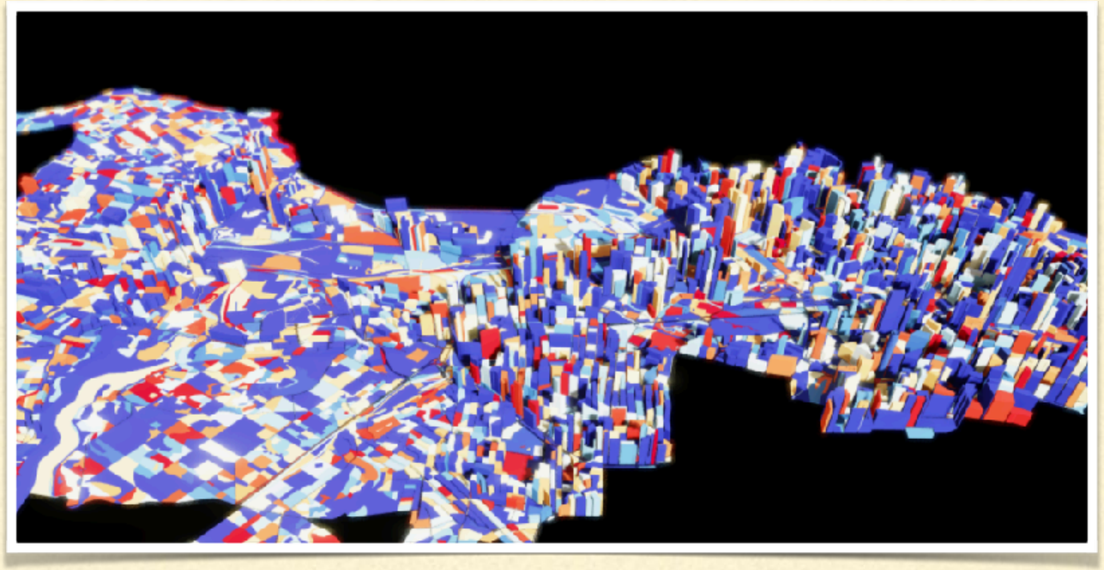
千万量级数据的前端展现

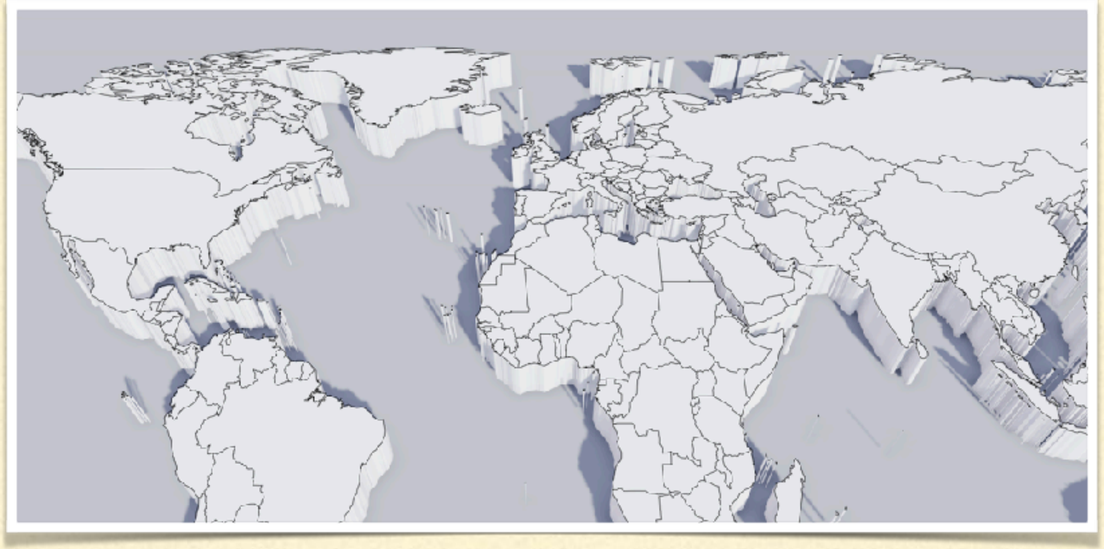


通过增量渲染技术（4.0+），配合各种细致的优化，ECharts 能够展现千万级的数据量，并且在这个数据量级依然能够进行流畅的缩放平移等交互。

几千万的地理坐标数据就算使用二进制存储也要占上百 MB 的空间。因此 ECharts 同时提供了对流加载（4.0+）的支持，你可以使用 WebSocket 或者对数据分块后加载，加载多少渲染多少！不需要漫长地等待所有数据加载完再进行绘制。


10000000 GPS Points





互联网的阴暗后厨

如何搜集用户位置数据?

GPS定位

基站定位

WiFi定位

IP定位

室内定位

谢谢！