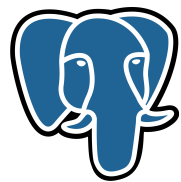


多维向量搜索

(图像识别、相似向量扩选、向量范围搜索)



阿里云
digoal

目录

- CUBE
- IMGSMR
- PASE
- 应用场景介绍

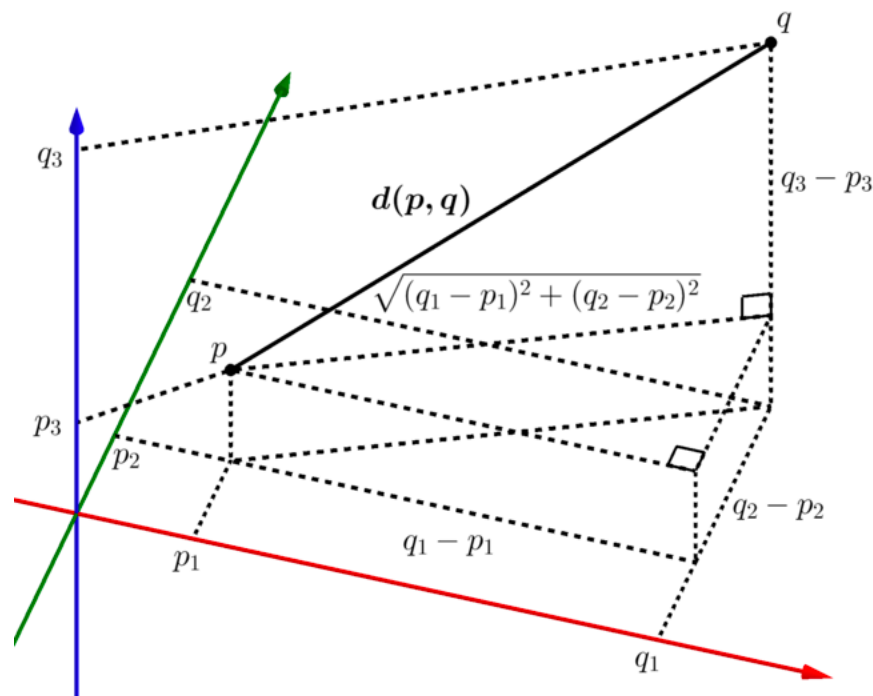
cube – 所有版本支持

- <https://www.postgresql.org/docs/current/cube.html>
- n维点: $(x1, x2, \dots, xn)$
- n维cube(采用2个对角point表示): $[(x1, \dots, xn), (y1, \dots, yn)]$
- 向量距离排序 (GiST索引) :

<code>a <-> b</code>	<code>float8</code>	Euclidean distance between a and b.
<code>a <#> b</code>	<code>float8</code>	Taxicab (L-1 metric) distance between a and b.
<code>a <=> b</code>	<code>float8</code>	Chebyshev (L-inf metric) distance between a and b.

Euclidean distance

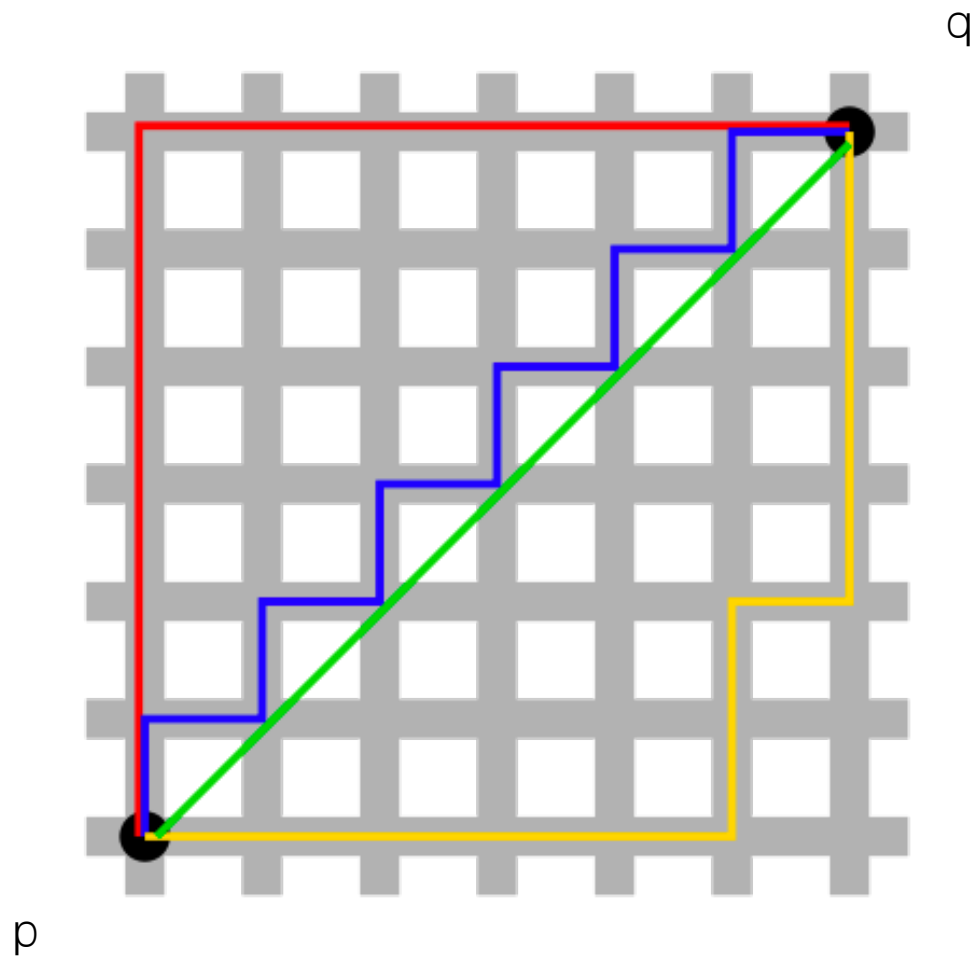
p与q的距离:



Taxicab distance


p与q的距离: 12

绿色为欧式距离 $\sqrt{12}$



Chebyshev distance

棋盘距离

	a	b	c	d	e	f	g	h	
8	5	4	3	2	2	2	2	2	8
7	5	4	3	2	1	1	1	2	7
6	5	4	3	2	1		1	2	6
5	5	4	3	2	1	1	1	2	5
4	5	4	3	2	2	2	2	2	4
3	5	4	3	3	3	3	3	3	3
2	5	4	4	4	4	4	4	4	2
1	5	5	5	5	5	5	5	5	1
	a	b	c	d	e	f	g	h	

例子

https://github.com/digoal/blog/blob/master/201810/20181011_01.md

1、创建插件

```
create extension cube;
```

2、创建测试表

```
create table tt (id int , c1 cube);
```

3、创建GIST索引

```
create index idx_tt_1 on tt using gist(c1);
```

4、创建生成随机CUBE的函数

```
create or replace function gen_rand_cube(int,int) returns cube as $$
```

```
select ('('||string_agg((random()*$1)::int::text, ',')||')')::cube from generate_series(1,$2);
```

```
$$ language sql strict;
```

```
db1=> select gen_rand_cube(10000,16); -- 生成16维向量，每一维度取值范围0-10000
```

```
gen_rand_cube
```

```
(6671, 1730, 2177, 4208, 2613, 4877, 3942, 4402, 244, 9945, 2581, 3640, 2384, 8457, 9126, 1102)
```

```
(1 row)
```

例子

6、写入测试数据（最多100个维度）

```
insert into tt select id, gen_rand_cube(10000, 16) from generate_series(1,1000000) t(id);
```

7、通过单个特征值CUBE查询相似人群，以点搜群 (<#> <=> <->)

```
select * from tt order by
```

```
c1 <=> '(6671, 1730, 2177, 4208, 2613, 4877, 3942, 4402, 244, 9945, 2581, 3640, 2384, 8457, 9126, 1102)' limit 10; -- 个体搜群体
```

执行计划:

QUERY PLAN

Limit (cost=0.41..1.30 rows=10 width=148)

-> Index Scan using idx_tt_1 on tt (cost=0.41..89127.71 rows=1000000 width=148)

Order By: (c1 <=> '(6671, 1730, 2177, 4208, 2613, 4877, 3942, 4402, 244, 9945, 2581, 3640, 2384, 8457, 9126, 1102)'::cube)

(3 rows)

例子

8、cube范围圈选人群

```
select * from tt where
```

```
'(7079, 3124, 1165, 4973, 3042, 6319, 5866, 3759, 1667, 7551, 3949, 3373, 4504, 9673, 9808, 199),
```

```
(17079, 13124, 11165, 14973, 11042, 16319, 15866, 3759, 1667, 17551, 13949, 13373, 14504, 19673, 19808, 1199)
```

```
:::cube
```

```
@> c1;
```

执行计划:

```
Index Scan using idx_tt_1 on tt (cost=0.41..1139.91 rows=1000 width=140)
```

```
Index Cond: (c1 <@ '(7079, 3124, 1165, 4973, 3042, 6319, 5866, 3759, 1667, 7551, 3949, 3373, 4504, 9673, 9808, 199),(17079, 13124, 11165, 14973, 11042, 16319, 15866, 3759, 1667, 17551, 13949,
```

```
13373, 14504, 19673, 19808, 1199):::cube)
```

```
(2 rows)
```

例子

9、通过多个特征值CUBE查询相似人群，以群搜群 (<#> <=> <->)

```
select * from tt order by c1 <=>
```

```
'(7079, 3124, 1165, 4973, 3042, 6319, 5866, 3759, 1667, 7551, 3949, 3373, 4504, 9673, 9808, 199),
```

```
(17079, 13124, 11165, 14973, 11042, 16319, 15866, 3759, 1667, 17551, 13949, 13373, 14504, 19673, 19808, 1199)
```

```
'::cube
```

```
limit 10; -- 群体搜群体
```

执行计划：

```
Limit (cost=0.41..1.30 rows=10 width=148)
```

```
-> Index Scan using idx_tt_1 on tt (cost=0.41..89127.71 rows=1000000 width=148)
```

```
    Order By: (c1 <=> '(7079, 3124, 1165, 4973, 3042, 6319, 5866, 3759, 1667, 7551, 3949, 3373, 4504, 9673, 9808, 199),(17079, 13124, 11165, 14973, 11042, 16319, 15866, 3759, 1667, 17551, 13949, 13373, 14504, 19673, 19808, 1199)::cube)
```

```
(3 rows)
```

目录

- CUBE
- IMGSMR
- PASE
- 应用场景介绍

imgsmr

- https://github.com/digoal/blog/blob/master/201809/20180904_02.md
- 与cube不一样的地方
 - imgsmr支持图像特征值转换
 - imgsmr内部使用float4存储每个维度的值

例子

```
create extension imgsmlr;
```

```
create table t_img_bytea (id int primary key, vid int, pic bytea); -- 原始表, 二进制存储原始图像
```

```
create table t_img_sig (id int primary key, vid int, sig signature, pat pattern); -- 特征表
```

```
create index idx_t_img_sig_1 on t_img_sig using gist(sig); -- 特征索引
```

```
insert into t_img_bytea values (.....); -- 写入原始图像
```

```
insert into t_img_sig select id, vid, png2pattern(pic), pattern2signature(png2pattern(pic)) from t_img_bytea; -- 转换为特征
```

```
select * from t_img_sig order by sig <-> pattern2signature(png2pattern('用户上传的照片的二进制')) limit 1; -- 根据图像搜索
```

目录

- CUBE
- IMGSMR
- PASE
- 应用场景介绍

pase – rds pg 11支持，即将覆盖所有主流版本

https://github.com/digoal/blog/blob/master/201912/20191219_02.md

https://help.aliyun.com/document_detail/147837.html

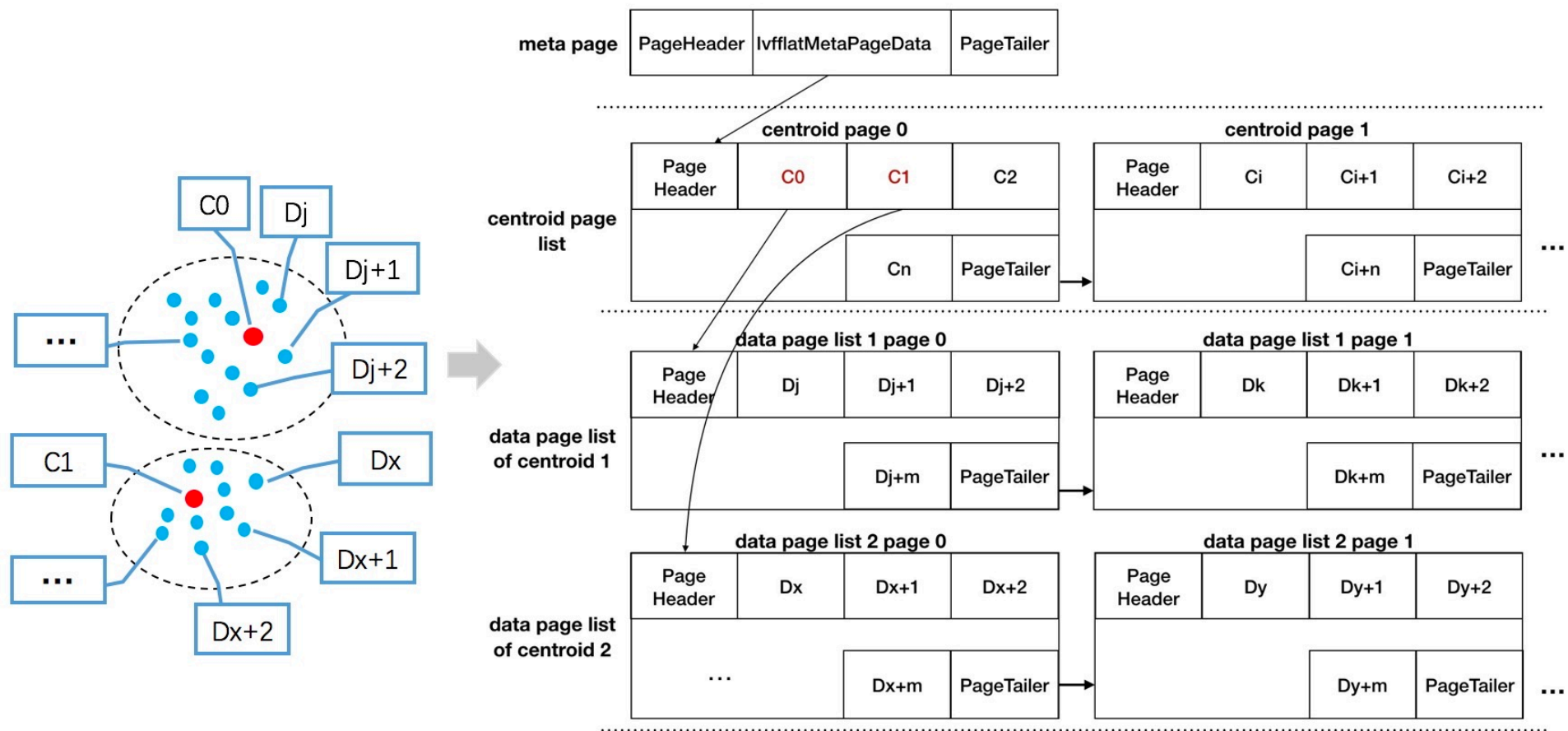
与cube,imgsmlr的差异

- 支持新增2种索引接口(ivfflat, hnsw)

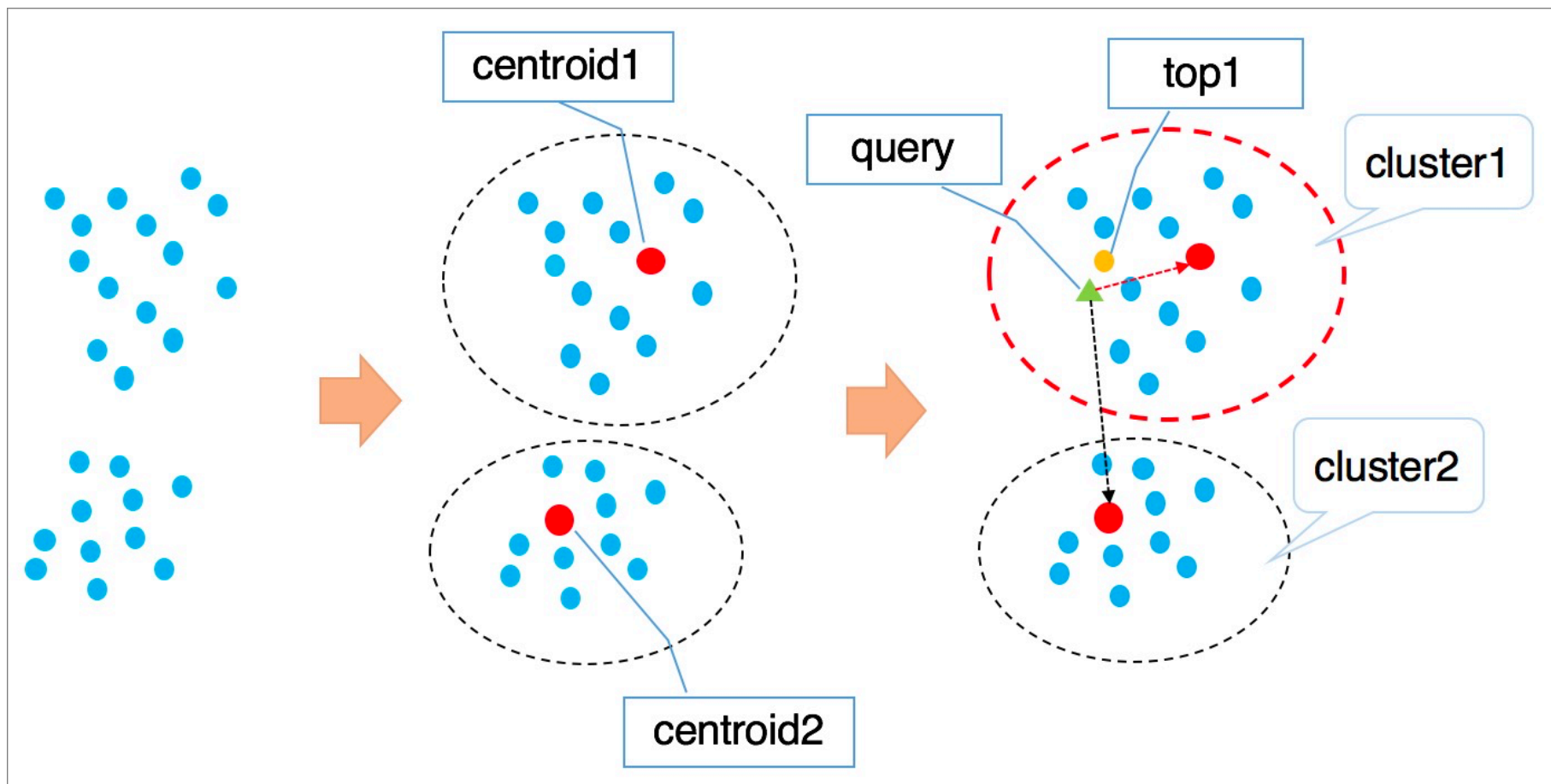
- 内部采用float4存储每个维度value

- 查询性能更好

ivfflat 索引存储结构介绍



ivfflat搜索介绍



- **算法流程说明：**

- 1、高维空间中的点基于隐形的聚类属性，按照kmeans等聚类算法对向量进行聚类处理，使得每个类簇有一个中心点。
- 2、检索向量时首先遍历计算所有类簇的中心点，找到与目标向量最近的n个类簇中心。
- 3、遍历计算n个类簇中心所在聚类中的所有元素，经过全局排序得到距离最近的k个向量。

- **说明**

- 在查询类簇中心点时，会自动排除远离的类簇，加速查询过程，但是无法保证最优的前k个向量全部在这n个类簇中，因此会有精度损失。您可以通过类簇个数n来控制IVFFlat算法的准确性，n值越大，算法精度越高，但计算量会越大。
- IVFFlat和IVFADC[2]的第一阶段完全一样，主要区别是第二阶段计算。IVFADC通过积量化来避免遍历计算，但是会导致精度损失，而IVFFlat是暴力计算，避免精度损失，并且计算量可控。

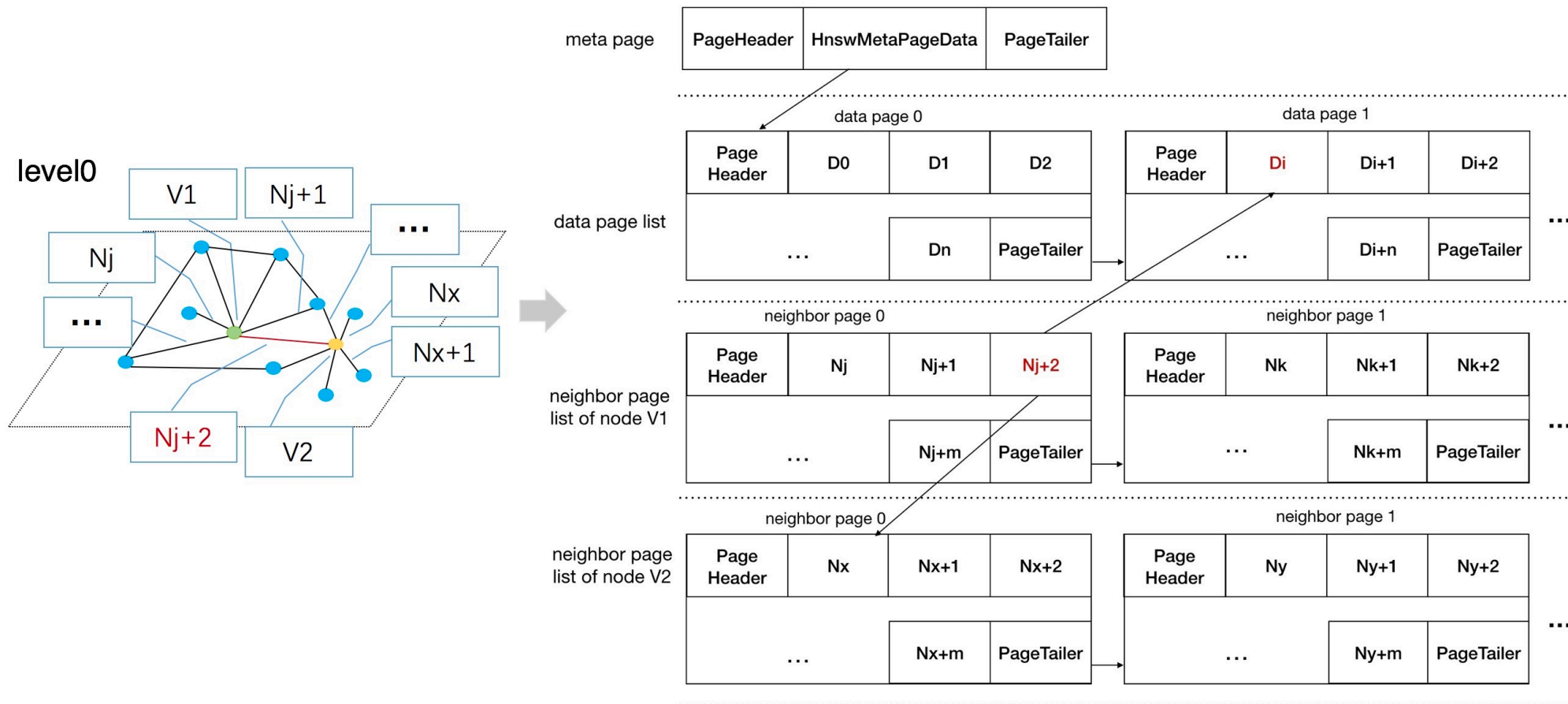
ivfflat索引|参数介绍

```
CREATE INDEX ${index_name} ON ${TABLE_NAME}
USING
pase_ivfflat(vector)
WITH
(clustering_type = 0, distance_type = 0, dimension = 256,
base64_encoded = 1, clustering_params =
"/data/centroid_path/centroids.v45.20190802");
```

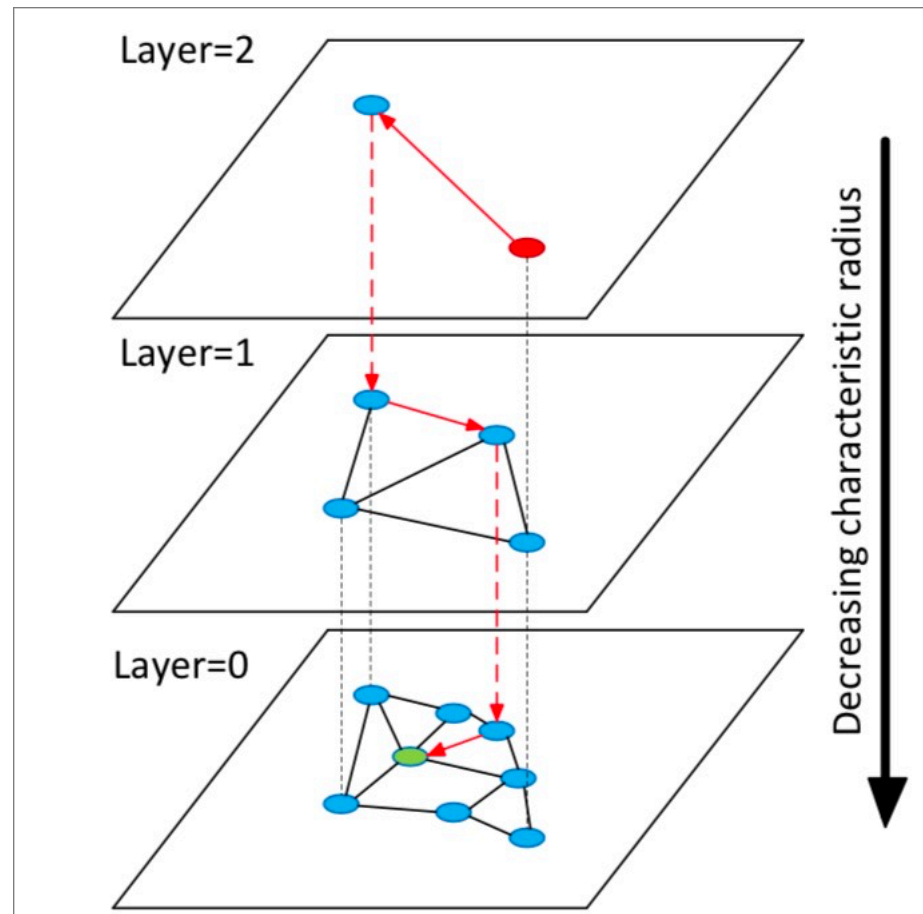
说明：

- 这一步100万数据耗时大概7分钟，其中index_name随意起，不重复即可。
- clustering_type: 0, 表示外部聚类，加载外部提供的中心点；1, 内部聚类。
- distance_type: 0, 欧式距离；1, 内积；2, 余弦。目前暂时只支持欧式距离，对于归一化的向量，余弦值的序和欧氏距离的序是反序关系。
 - （目前索引build索引时：仅支持欧式距离。默认认为数据是归一化(向量原点归一)处理的。查询时（计算distance）支持三种距离算法。排序只支持欧式距离，因为索引顺序是固定的（因为建索引时认为数据是归一化的））
- dimension: 向量维度。
- base64_encoded: 数据输出输出格式是否采用base64编码。
- clustering_params:
 - 1、对于外部聚类，该项配置为中心点文件路径。
 - 2、对于内部聚类，该项配置为聚类参数，格式为：clustering_sample_ratio,k,
 - clustering_sample_ratio为库内数据用作聚类的采样比例，范围(0,1000]内的整数，
 - k为聚类中心数。
 - ps: 中心点个数可以理解为桶的个数，向量点根据它离中心点的距离被聚类到对应的桶。
 - 当使用内部聚类时，clustering_sample_ratio表示从原始表数据的千分之多少来构建k个中心点。然后生成这些中心点的k个桶。

hnsw 索引存储结构介绍



hnsw搜索介绍



- **算法流程说明：**

- 1、构造多层图，每层图都是下层图的一个缩略，同时构成下层图的跳表，类似高速公路。
- 2、从顶层随机选中一个点开始查询。
- 3、第一次搜索其邻居点，把它们按距离目标的远近顺序存储在定长的动态列表中，以后每一次查找，依次取出动态列表中的点，搜索其邻居点，再把这些新探索的邻居点插入动态列表，每次插入动态列表需要重新排序，保留前k个。如果列表有变化则继续查找，不断迭代直至达到稳态，然后以动态列表中的第一个点作为下一层的入口点，进入下一层。循环执行第3步，直到进入最底层。

- **说明：**

- HNSW算法是在NSW算法的单层构图的基础上构造多层图，在图中进行最近邻查找，可以实现比聚类算法更高的查询加速比。

hnsw索引参数介绍

```
CREATE INDEX ${index_name} ON  ${TABLE_NAME}
USING
palaemon_hnsw(vector)
WITH
(dim = 256, base_nb_num = 32, ef_build = 80, ef_search
= 100, base64_encoded = 1);
```

说明：

- `base_nb_num`：邻居数，第0层 $2 * base_nb_num$ ，高层为`base_nb_num`。要达到order by limit N的最好效果时，`base_nb_num`最好大于N
- `ef_build`： build索引的时候第0层的堆长度（构图时，0层邻居点来构图，上层都是1个点），越大效果越好，build越慢。不是最终索引存的内容，只是build时构图的一个参数。 要达到较好的构图效果，建议`ef_build`大于`base_nb_num`，否则要到下层去提取满足`base_nb_num`个数的邻居点，精度就没有那么高了。
- `ef_search`： 查询的堆长度，越大效果越好，search性能越差，可在查询时指定，该处为默认值。查询时从上层往下查，查到第一层为止（第0层存的是heap table ctid(s)），返回第0层的ctids。 如果要保证order by limit N可以返回N条记录，`ef_search`要大于N
- `base64_encoded`： 数据输入输出格式是否采用base64编码格式。搜索时从上层往下搜索，一直搜索到第1层，返回第0层的ctids。

例子

创建测试表

```
create table if not exists t_pase_80(  
    id serial PRIMARY KEY,  
    vec float4[]  
);
```

创建生成随机float4数组的函数

```
create or replace function gen_float4_arr(int,int) returns float4[] as $$  
    select array_agg(trunc(random()*$1)::float4) from generate_series(1,$2);  
$$ language sql strict volatile;
```


例子

写入100万随机80维向量,

```
insert into t_pase_80 (vec) select gen_float4_arr(10000,80) from generate_series(1,1000000);
```

创建ivfflat索引。

```
CREATE INDEX idx_t_pase_80_2 ON t_pase_80
```

```
USING
```

```
    pase_ivfflat(vec)
```

```
WITH
```

```
(clustering_type = 1, distance_type = 0, dimension = 80, clustering_params = "100,1001");
```

采样记录数：100万记录乘以100/1000=10万。 中心点：生成1001个中心点

NOTICE: vector dimension is huge, parameter (clustering_sample_ratio) should be set to ensure the clustering count lower than 983040

NOTICE: parse clustering parameters succeed, clustering_sample_ratio[100], k[1001]

例子

查询一条真实记录，略微修改几个维度

```
select * from t_pase_80 limit 1;
```

使用ivfflat索引查询 (<#>)

```
SELECT id,
```

```
vec <#>
```

```
'1841,9512,8870,4345,3829,9005,738,2568,2564,6642,2455,7807,1666,4880,9195,6239,788,2804,301,6808,8182,1271,9446,1324,7230,78  
68,3294,9092,4189,6227,2400,6029,5739,1271,375,9568,277,1114,2137,2841,7756,4593,649,9422,9473,9844,5662,262,2650,5964,7071,8  
31,7235,6518,2156,4466,4386,5450,3558,8576,1677,5959,4606,7417,7230,4981,6985,7508,6095,9123,349,3852,3716,998,3275,3190,843,  
8938,3462,3499:0:0'::pase as distance
```

```
FROM t_pase_80
```

```
ORDER BY
```

```
vec <#>
```

```
'1841,9512,8870,4345,3829,9005,738,2568,2564,6642,2455,7807,1666,4880,9195,6239,788,2804,301,6808,8182,1271,9446,1324,7230,78  
68,3294,9092,4189,6227,2400,6029,5739,1271,375,9568,277,1114,2137,2841,7756,4593,649,9422,9473,9844,5662,262,2650,5964,7071,8  
31,7235,6518,2156,4466,4386,5450,3558,8576,1677,5959,4606,7417,7230,4981,6985,7508,6095,9123,349,3852,3716,998,3275,3190,843,  
8938,3462,3499:0:0'::pase
```

```
LIMIT 10;
```

例子

id	distance
1	139
620286	6.78452e+08
365838	1.62702e+09
365885	1.667e+09
988412	1.57742e+09
17530	1.58652e+09
821096	1.57582e+09
820902	1.57803e+09
128421	1.57324e+09
127295	1.80574e+09

(10 rows)

Time: 4.523 ms

目录

- CUBE
- IMGSMR
- PASE
- 应用场景介绍

多维向量搜索应用场景

- 图像识别、人脸识别、以图搜图
- 精准广告营销系统：相似向量人群扩选、
- 精准广告营销系统：圈选向量取值范围人群

参考资料

- 案例
- MySQL手册
 - <https://www.mysqltutorial.org/>
 - <https://dev.mysql.com/doc/refman/8.0/en/>
- PG 管理、开发规范
 - https://github.com/digoal/blog/blob/master/201609/20160926_01.md
- PG手册
 - <https://www.postgresql.org/docs/current/index.html>
 - <https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-vs-mysql/>
- GIS手册
 - <http://postgis.net/docs/manual-3.0/>

一期开课计划(PG+MySQL联合方案)

- - 2019.12.30 19:30 RDS PG产品概览， 如何与MySQL结合使用
- - 2019.12.31 19:30 如何连接PG， GUI， CLI的使用
- - 2020.1.3 19:30 如何压测PG数据库、如何瞬间构造海量测试数据
- - 2020.1.6 19:30 MySQL与PG对比学习(面向开发者)
- - 2020.1.7 19:30 如何将MySQL数据同步到PG (DTS)
- - 2020.1.8 19:30 PG外部表妙用 - mysql_fdw, oss_fdw (直接读写MySQL数据、冷热分离)
- - 2020.1.9 19:30 PG应用场景介绍 - 并行计算， 实时分析
- - 2020.1.10 19:30 PG应用场景介绍 - GIS
- - 2020.1.13 19:30 PG应用场景介绍 - 用户画像、实时营销系统
- - 2020.1.14 19:30 PG应用场景介绍 - 多维搜索
- - 2020.1.15 19:30 PG应用场景介绍 - 向量计算、图像搜索
- - 2020.1.16 19:30 PG应用场景介绍 - 全文检索、模糊查询
- - 2020.1.17 19:30 PG 数据分析语法介绍
- - 2020.1.18 19:30 PG 更多功能了解：扩展语法、索引、类型、存储过程与函数。如何加入PG技术社群

本课程习题

- 多维向量如何表达？内部使用什么存储？
- 多维向量支持哪几种索引？
- 多维向量支持哪几种距离计算方法？
- 多维向量广泛应用于哪些场景？
- 1000万的多维向量，任意召回最相似的1000条大概需要多久？

技术社群



PG技术交流钉钉群(3600+人)

