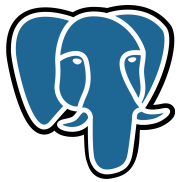
多维组合搜索

(任意字段组合查询)



阿里云

digoal

目录

- 任意字段组合查询
- 索引结构和原理
- 数据扫描方法
- 应用实践
  - 128个字段，任意字段组合搜索-gin,bitmapscan
  - 任意字段组合搜索-非字典化（column_prefix）gin\rum倒排搜索
  - 任意字段组合搜索-字典化gin\rum倒排搜索
  - 时空、数组、标量等多维度搜索(50倍提速vs ES)

任意字段组合查询

```
select ... from xx
where
x=? and
y=? and
z>=? or
(a=? and b=?)
... ...  -- 任意组合
order by c,d desc;
```

典型应用场景：

- ERP系统
- 搜索
- 拖拽式分析系统（任意维度过滤）

加速思路

- 扫描最少的索引块
- 扫描最少的表数据库块
- 过滤最少的不符合条件的记录
- 尽量避免对大量记录进行显示排序

- 核心
- 索引数据结构设计
- 索引存储组织形式
- 表存储组织形式

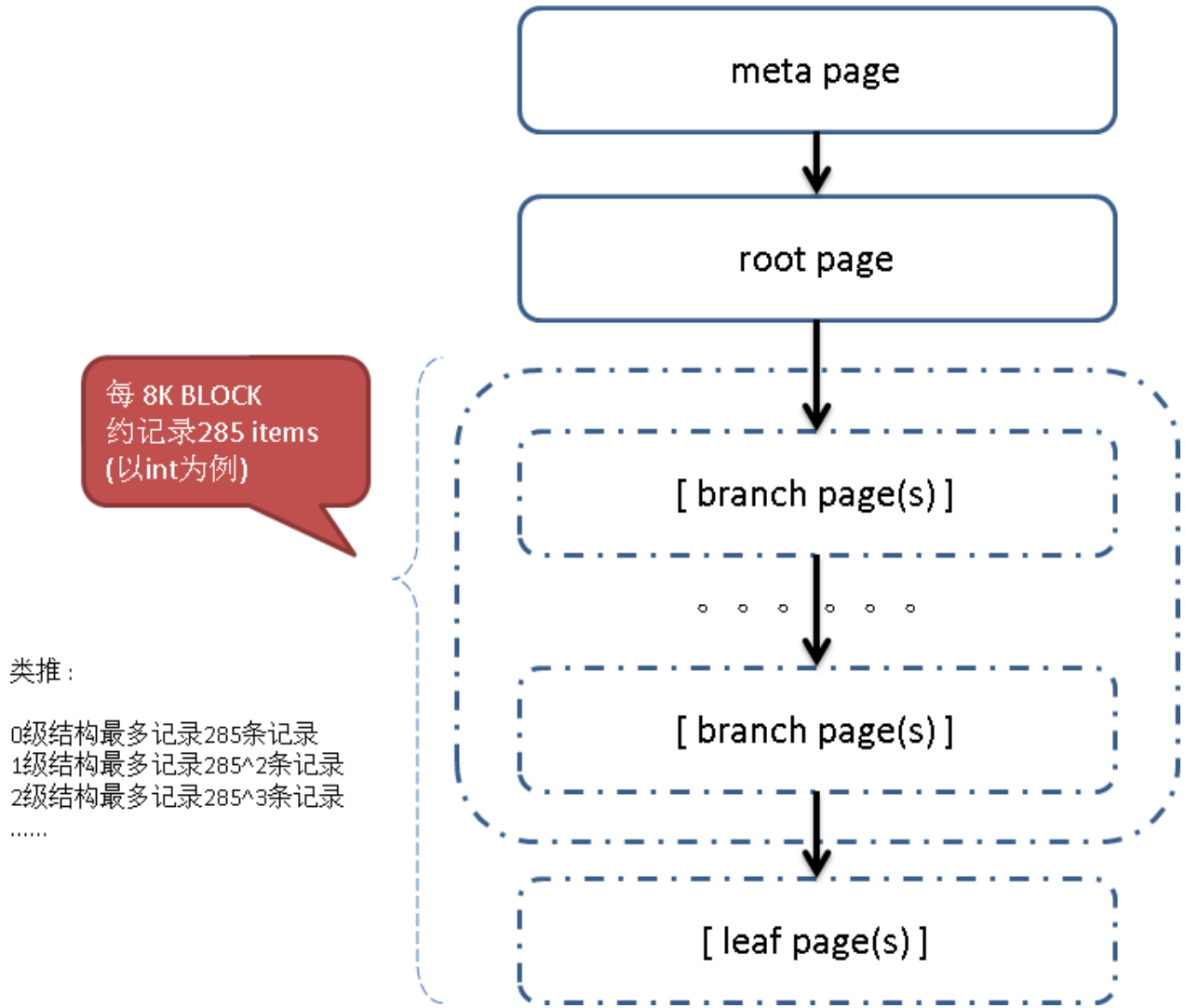
目录

- 任意字段组合查询
- 索引结构和原理
- 数据扫描方法
- 应用实践
  - 128个字段，任意字段组合搜索-gin,bitmapscan
  - 任意字段组合搜索-非字典化（column_prefix）gin\rum倒排搜索
  - 任意字段组合搜索-字典化gin\rum倒排搜索
  - 时空、数组、标量等多维度搜索(50倍提速vs ES)

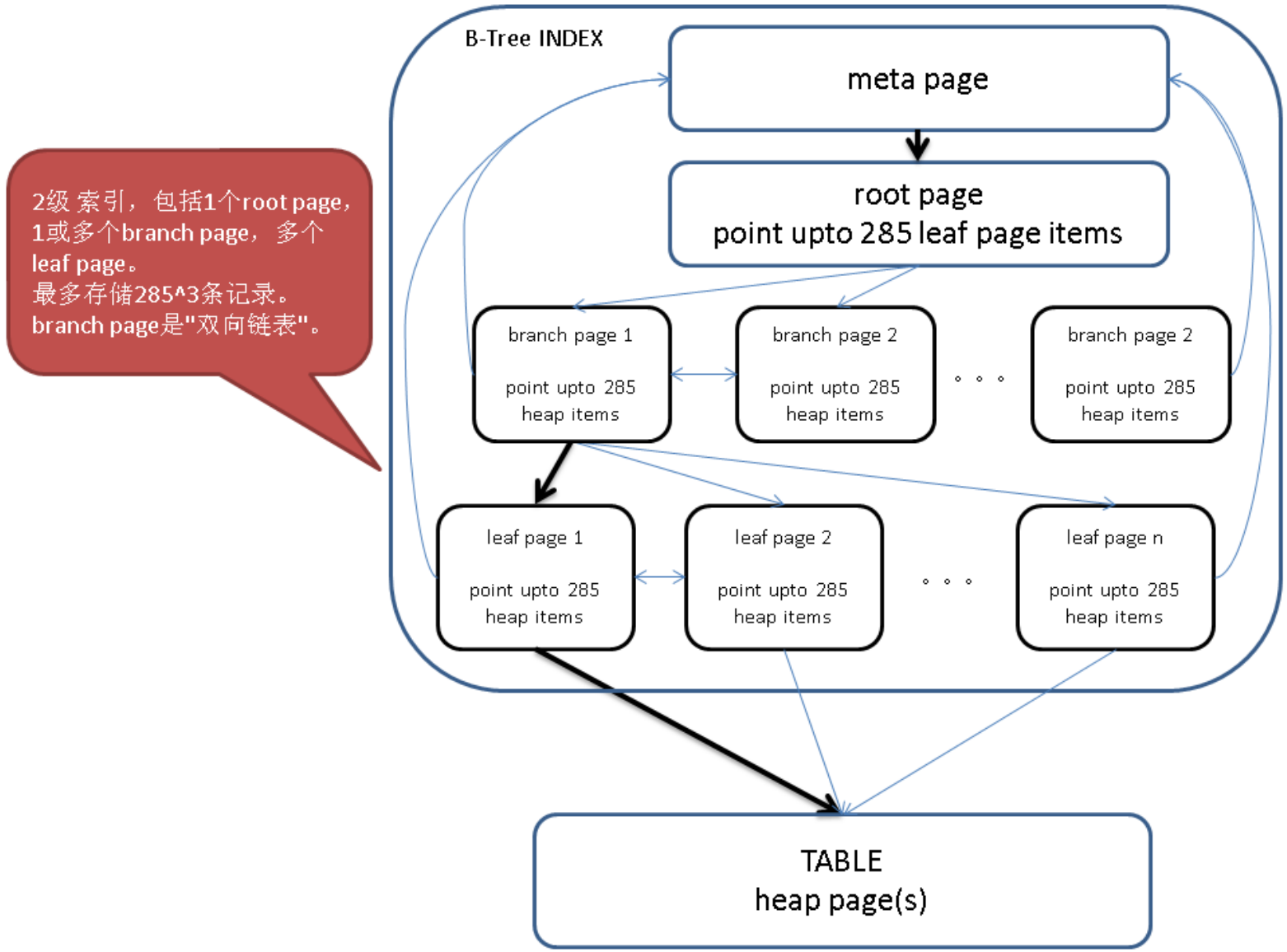
btree索引结构

- https://github.com/digoal/blog/blob/master/201605/20160528_01.md
- src/backend/access/nbtree/README

btree索引结构


meta page
root page
每 8K BLOCK
约记录285 items
(以int为例)
[branch page(s)]
[branch page(s)]
[leaf page(s)]
类推：
0级结构最多记录285条记录
1级结构最多记录285^2条记录
2级结构最多记录285^3条记录
.....

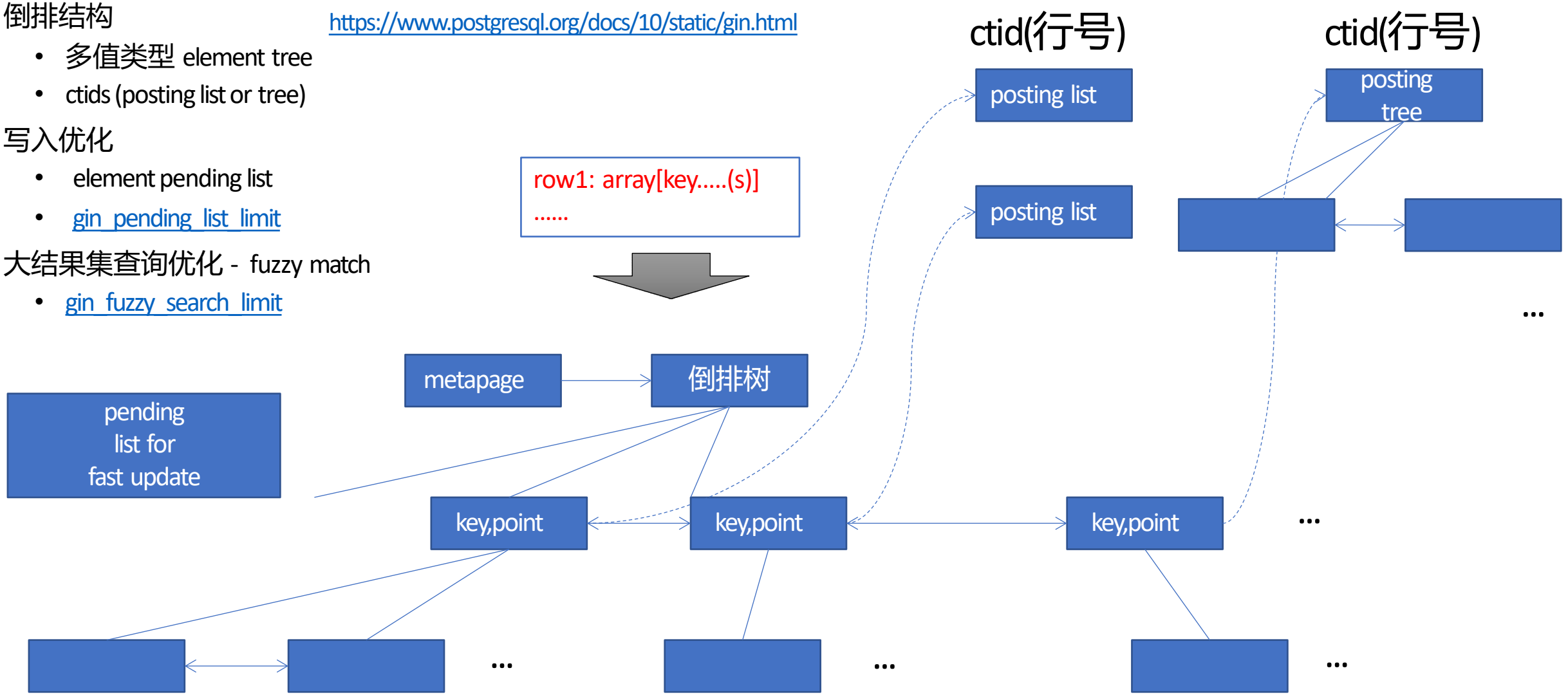
btree索引结构


2级 索引，包括1个root page，1或多个branch page，多个leaf page。
最多存储285^3条记录。
branch page是"双向链表"。
B-Tree INDEX
meta page
root page
point upto 285 leaf page items
branch page 1
point upto 285
heap items
branch page 2
point upto 285
heap items
branch page 2
point upto 285
heap items
leaf page 1
point upto 285
heap items
leaf page 2
point upto 285
heap items
leaf page n
point upto 285
heap items
TABLE
heap page(s)

gin索引结构

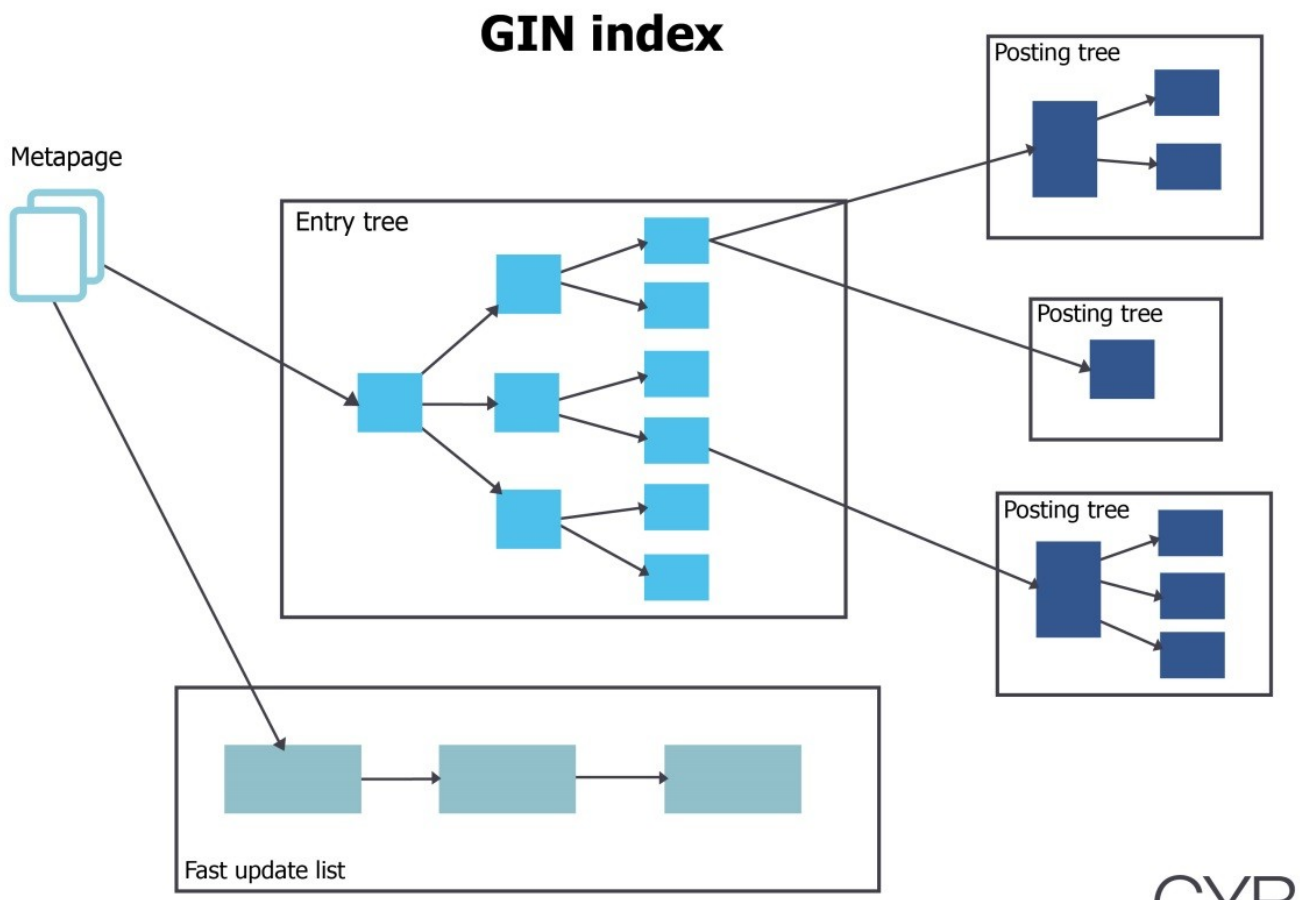
- src/backend/access/gin/README
- 倒排结构
  - 多值类型 element tree
  - ctids (posting list or tree)
- 写入优化
  - element pending list
  - gin_pending_list_limit
- 大结果集查询优化 - fuzzy match
  - gin_fuzzy_search_limit

https://www.postgresql.org/docs/10/static/gin.html



gin索引结构

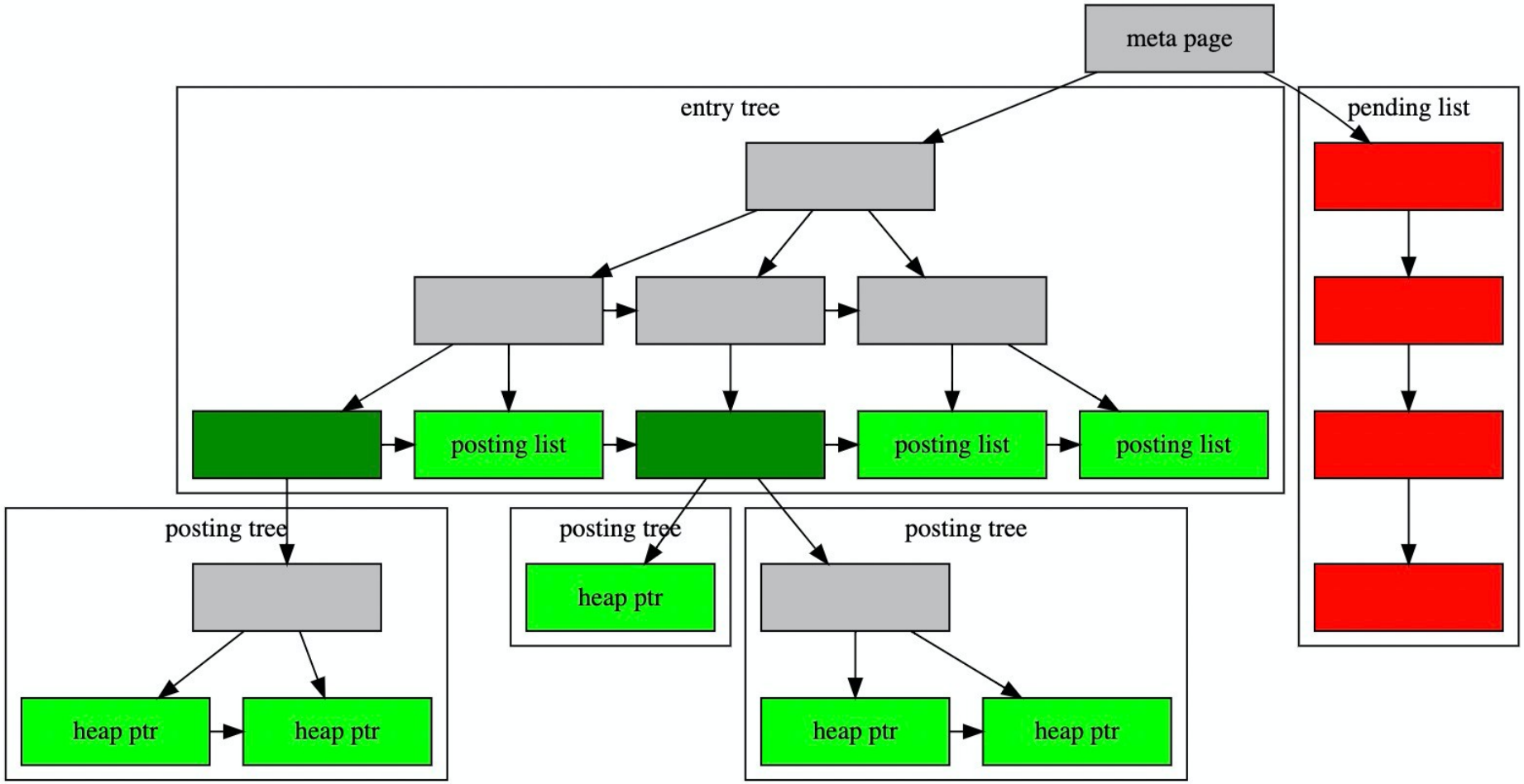
- https://www.cybertec-postgresq
- https://www.postgresql.org/docs/



gin索引结构

- https://www.cybertec-po
- https://www.postgresql.or

Figure 66.1. GIN Internals



`gin_page_opaque_info` returns information about a GIN index opaque area, like t

```
test=# SELECT * FROM gin_page_opaque_info(get_raw_page('gin_index', 2));
 rightlink | maxoff |         flags
-----------+--------+-----------------------
         5 |      0 | {data,leaf,compressed}
(1 row)
```

`gin_leafpage_items` returns information about the data stored in a GIN leaf page. For example:

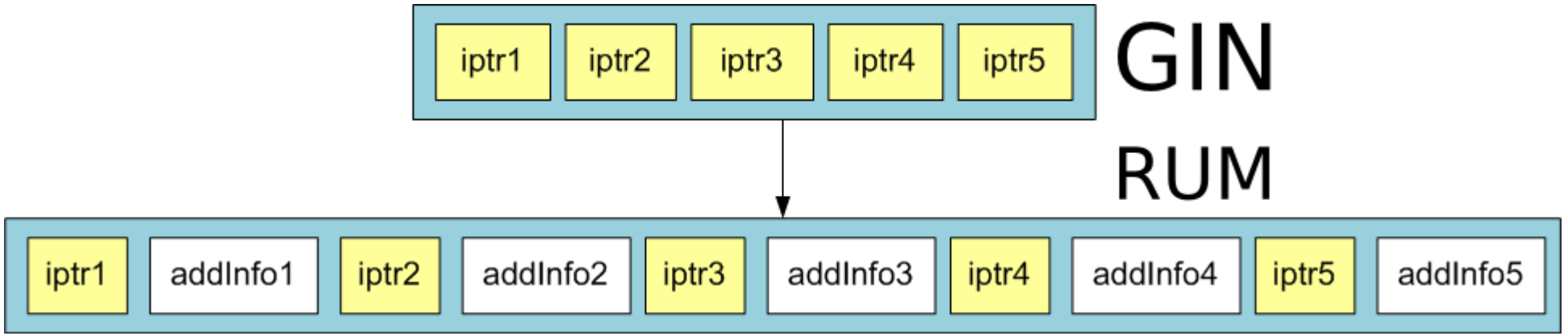
内窥GIN内容 https://www.postgres
1.11.7.31.7

```
test=# SELECT first_tid, nbytes, tids[0:5] AS some_tids
       FROM gin_leafpage_items(get_raw_page('gin_test_idx', 2));
 first_tid | nbytes |                          some_tids
-----------+--------+-------------------------------------------------------------
 (8,41)    |    244 | {"(8,41)","(8,43)","(8,44)","(8,45)","(8,46)"}
 (10,45)   |    248 | {"(10,45)","(10,46)","(10,47)","(10,48)","(10,49)"}
 (12,52)   |    248 | {"(12,52)","(12,53)","(12,54)","(12,55)","(12,56)"}
 (14,59)   |    320 | {"(14,59)","(14,60)","(14,61)","(14,62)","(14,63)"}
 (167,16)  |    376 | {"(167,16)","(167,17)","(167,18)","(167,19)","(167,20)"}
 (170,30)  |    376 | {"(170,30)","(170,31)","(170,32)","(170,33)","(170,34)"}
 (173,44)  |    197 | {"(173,44)","(173,45)","(173,46)","(173,47)","(173,48)"}
(7 rows)
```

rum索引结构

- https://github.com/postgrespro/rum
- https://github.com/digoal/blog/blob/master/201907/20190706_01.md

额外信息（例如attach column's value, tsvector's 出现次数等）

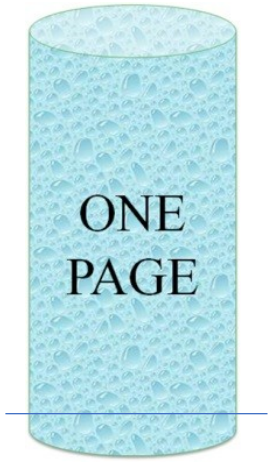


```
SELECT t, a <=> to_tsquery('english', 'beautiful | place') AS rank
    FROM test_rum
    WHERE a @@ to_tsquery('english', 'beautiful | place')
    ORDER BY a <=> to_tsquery('english', 'beautiful | place');
                t                 |   rank
----------------------------------+---------
 It looks like a beautiful place  | 8.22467
 The situation is most beautiful  | 16.4493
 It is a beautiful                | 16.4493
(3 rows)
```

```
CREATE INDEX tsts_idx ON tsts USING rum (t rum_tsvector_addon_ops, d)
    WITH (attach = 'd', to = 't');

Now we can execute the following queries:

EXPLAIN (costs off)
    SELECT id, d, d <=> '2016-05-16 14:21:25' FROM tsts WHERE t @@ 'wr&qh' ORDER BY d <=> '2016-05-16 14:21:25' LIMIT 5;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Limit
   ->  Index Scan using tsts_idx on tsts
         Index Cond: (t @@ '''wr'' & ''qh'''::tsquery)
         Order By: (d <=> 'Mon May 16 14:21:25 2016'::timestamp without time zone)
(4 rows)

SELECT id, d, d <=> '2016-05-16 14:21:25' FROM tsts WHERE t @@ 'wr&qh' ORDER BY d <=> '2016-05-16 14:21:25' LIMIT 5;
 id  |                d                |   ?column?
-----+---------------------------------+---------------
 355 | Mon May 16 14:21:22.326724 2016 |      2.673276
 354 | Mon May 16 13:21:22.326724 2016 |   3602.673276
 371 | Tue May 17 06:21:22.326724 2016 |  57597.326724
 406 | Wed May 18 17:21:22.326724 2016 | 183597.326724
 415 | Thu May 19 02:21:22.326724 2016 | 215997.326724
(5 rows)
```

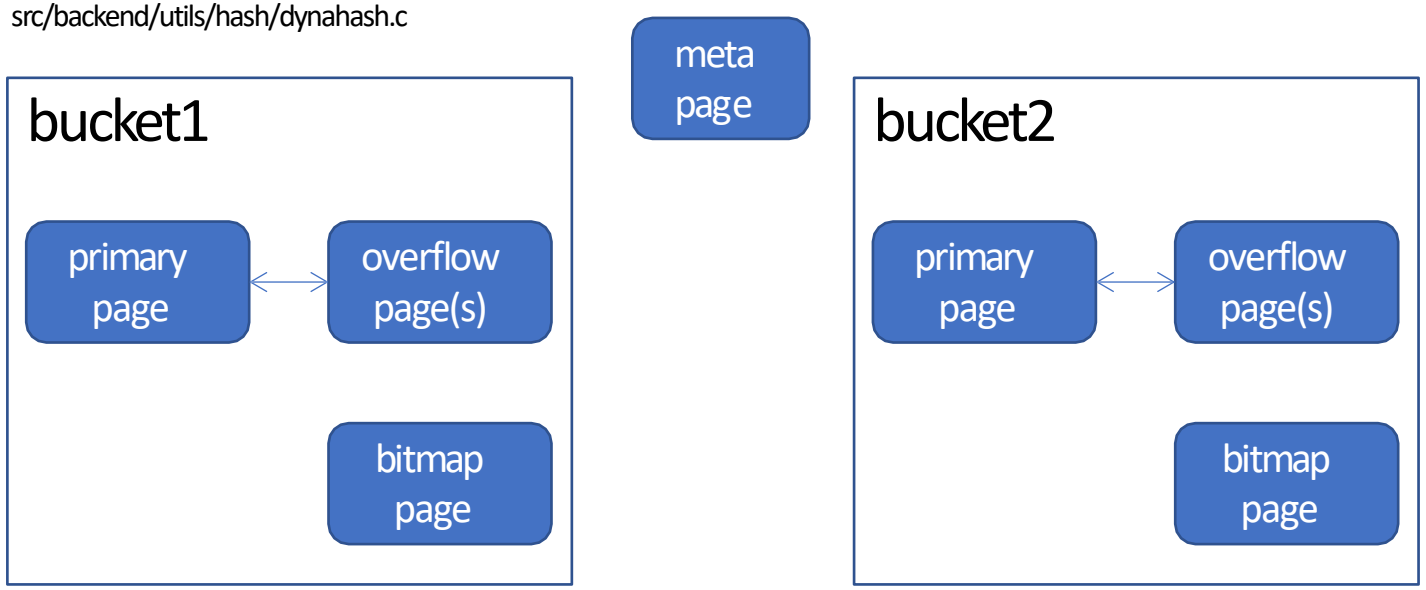
hash索引结构

- hash值转换，hash值映射到某个bucket。
- bucket数量为2的N次方。至少包括2个bucket。
- metapage，page zero。包括控制信息。
- 每个bucket内至少一个primary page。放不下时，增加overflow page。
- hash index支持长字符串。page内存储的是HASH VALUE。
- 每个page内，hash value有序存放，支持binary search. 跨page不保证有序。
- 分裂优化，增加bucket时，hash mapping会变化，需要分裂。2^n映射。有一定的优化策略
- (切成4个部分，增量进行split)。
- src/backend/access/hash/README
- src/backend/utils/hash/dynahash.c



special space

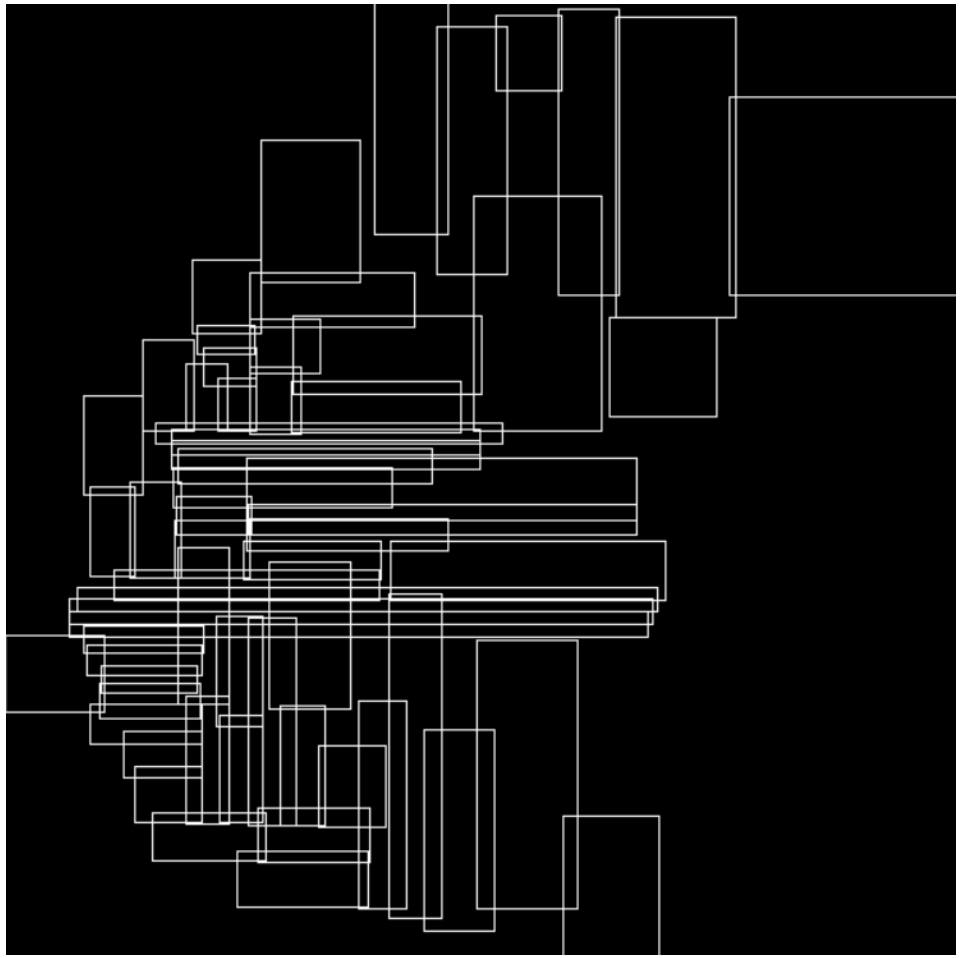
double link

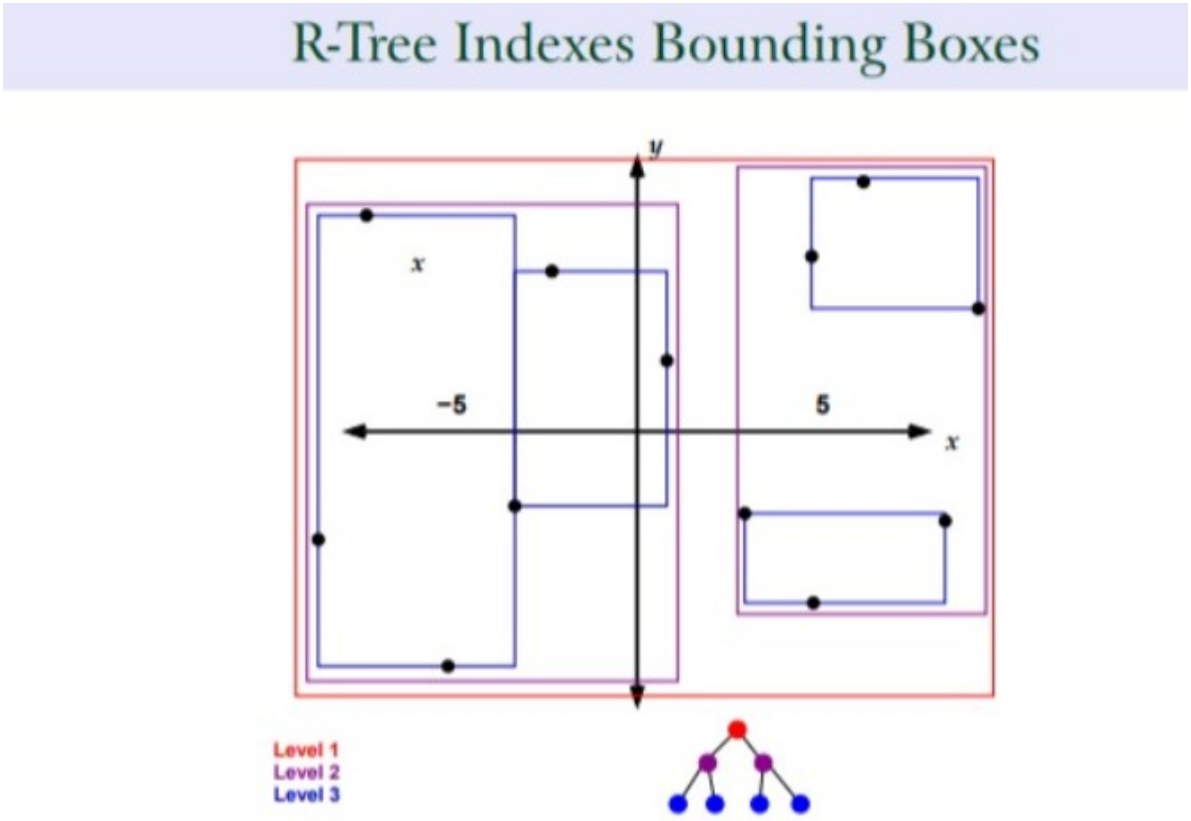


... ...

bitmap page:
标记overflow page 状态
(reuse,free)

gist索引结构





Geographic objects (lines, polygons) also can appear in r-tree indexes. based on their own bounding boxes.

spgist索引结构

- [空间分区]通用索引结构
- r-tree base on gist
- src/backend/access/gist/README
- src/backend/access/spgist/README
- https://github.com/digoal/blog/blob/master/201708/20170824_02.md
- https://github.com/digoal/blog/blob/master/201708/20170820_01.md
- https://github.com/digoal/blog/blob/master/201709/20170905_01.md
- https://github.com/digoal/blog/blob/master/201708/20170825_01.md

brin索引结构

- src/backend/access/brin/README
- 定义粒度
  - N个连续的块
- 索引字段值在连续N个块内的边界值
  - 普通边界
  - RANGE边界
  - 空间边界（BOUND BOX）
  - PostgreSQL 11 优化(分段 bound box)
    - https://github.com/digoal/blog/blob/master/201803/20180323_05.md

bloom索引结构

- bloom
  - https://www.postgresql.org/docs/devel/static/bloom.html
  - https://github.com/digoal/blog/blob/master/201605/20160523_01.md
  - https://en.wikipedia.org/wiki/Bloom_filter

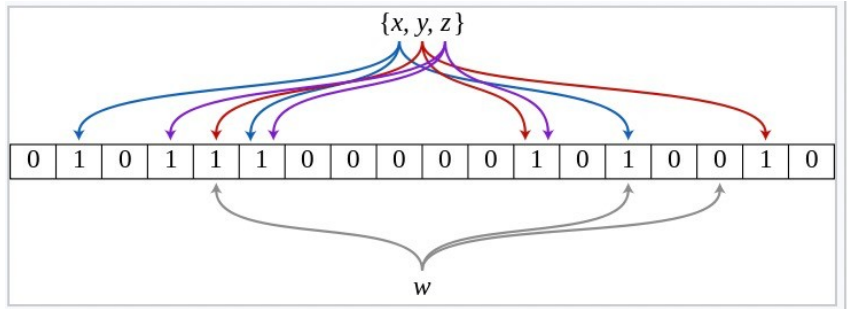
```
CREATE INDEX bloomidx ON tbloom USING bloom (i1,i2,i3)
       WITH (length=80, col1=2, col2=2, col3=4);
```

length, m值，即总共多少个bit位表示一个被索引的行。
Col1,col2,...，该列用几个bit表示，每个bit对应position是否设置为1由一个hash函数计算得到.
最多需要m个不同的hash函数，hash函数返回的值范围是0 ~ m-1 (即bit position).

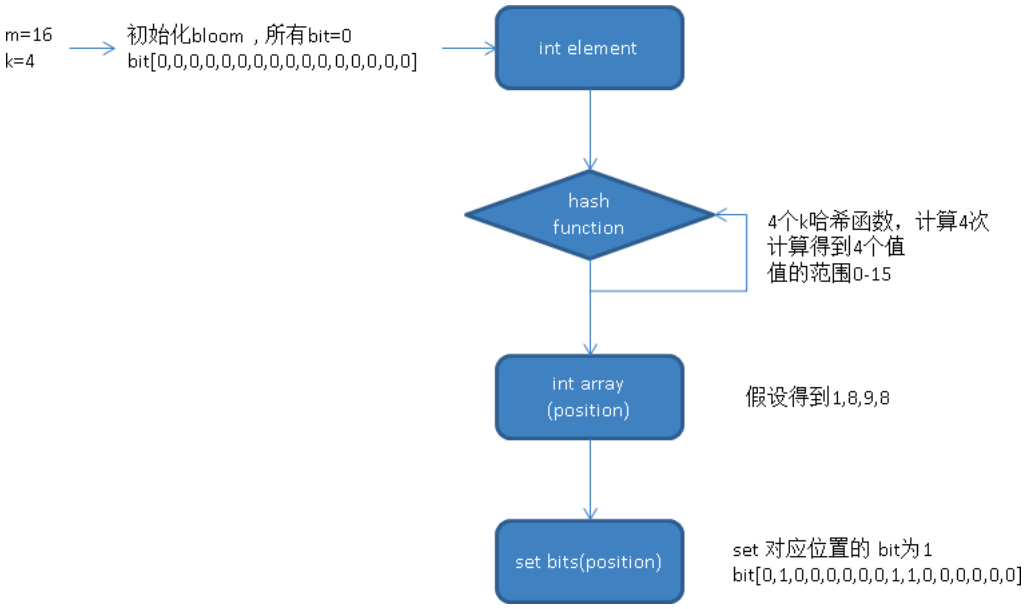
假设col1=3， 则索引的第一列需要3个hash函数来计算出3个bit position上的值是1或0.
假设m=10
i1='abc', hash1('abc')=3, hash2('abc')=5, hash3('abc')=9, 那么m=0001010001

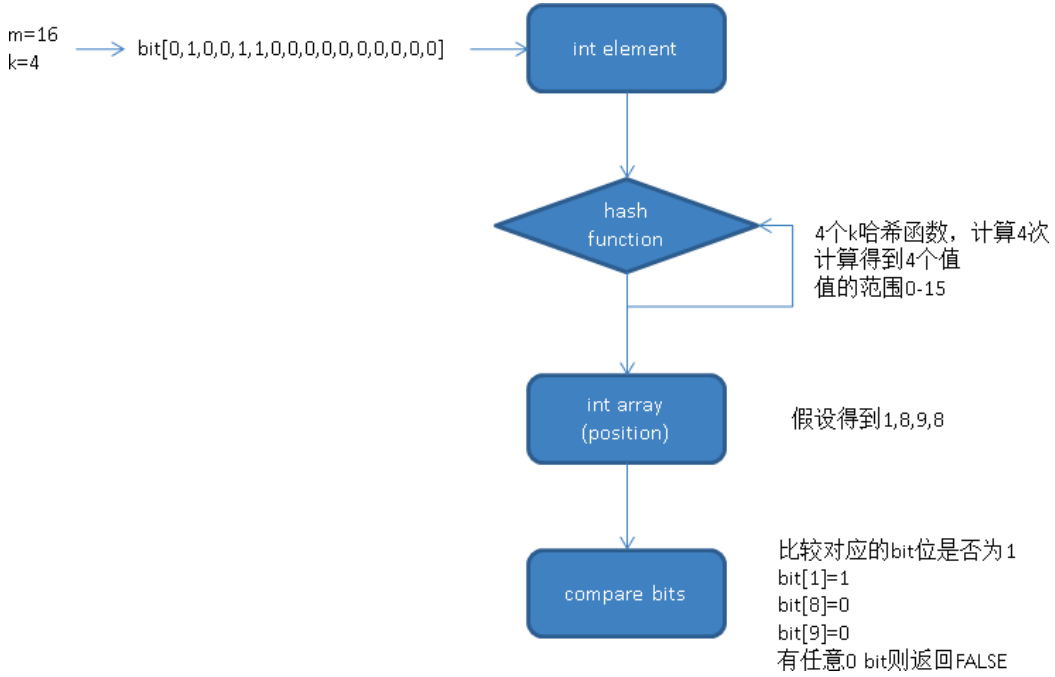
bloom索引结构

- bloom
  - https://www.postgresql.org/docs/devel/static/bloom.html
  - https://github.com/digoal/blog/blob/master/201605/20160523_01.md
  - https://en.wikipedia.org/wiki/Bloom_filter



An example of a Bloom filter, representing the set $\{x, y, z\}$. The colored arrows show the positions in the bit array that each set element is mapped to. The element w is not in the set $\{x, y, z\}$, because it hashes to one bit-array position containing 0. For this figure, $m = 18$ and $k = 3$.





其他索引结构

- zombodb
  - https://github.com/zombodb/zombodb

  其他索引讲解文档
  https://github.com/digoal/blog/blob/master/201908/20190816_01.md

目录

- 任意字段组合查询
- 索引结构和原理
- 数据扫描方法
- 应用实践
  - 128个字段，任意字段组合搜索-gin,bitmapscan
  - 任意字段组合搜索-非字典化（column_prefix）gin\rum倒排搜索
  - 任意字段组合搜索-字典化gin\rum倒排搜索
  - 时空、数组、标量等多维度搜索(50倍提速vs ES)

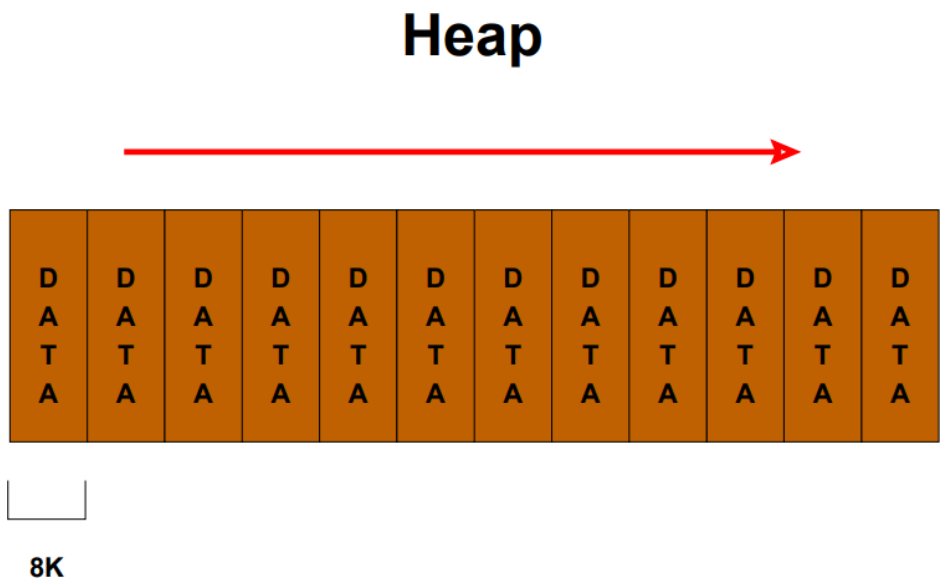
扫描方法介绍

- seqscan
- index only scan
- index scan
- bitmap scan
- ctid scan

seqscan

从0号数据块开始扫

表大小超过SB/4, 加TAG BAS_BULKREAD ,优先淘汰

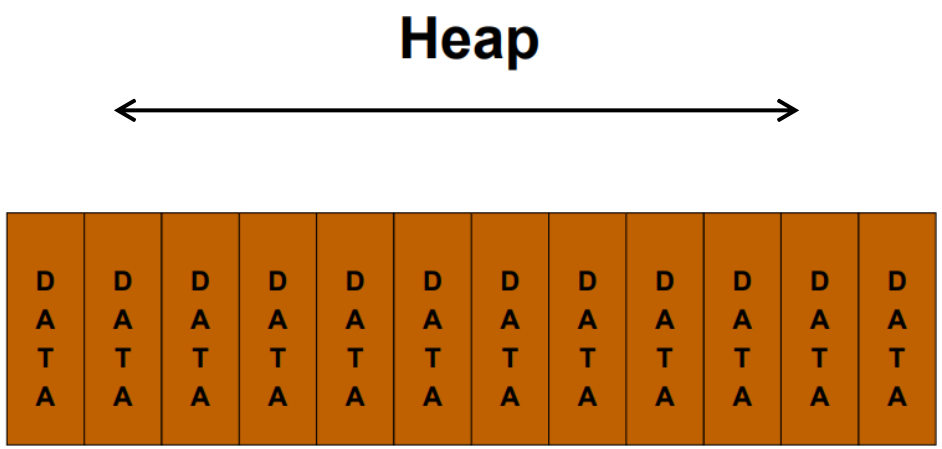


分批处理，并不会把shared buffer塞满

seqscan+synchronize_seqscans=on

可能从中间开始扫，
WRAP

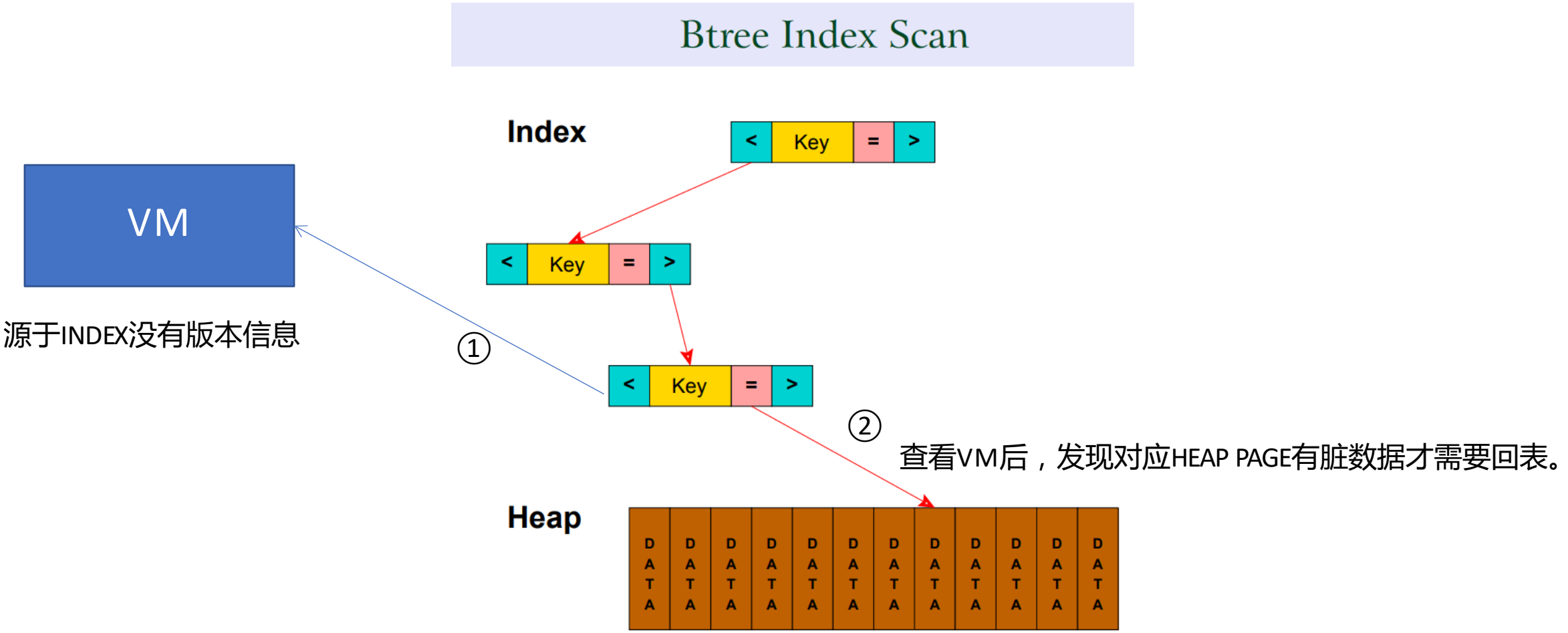
表大小超过SB/4, 加TAG
BAS_BULKREAD
,优先淘汰



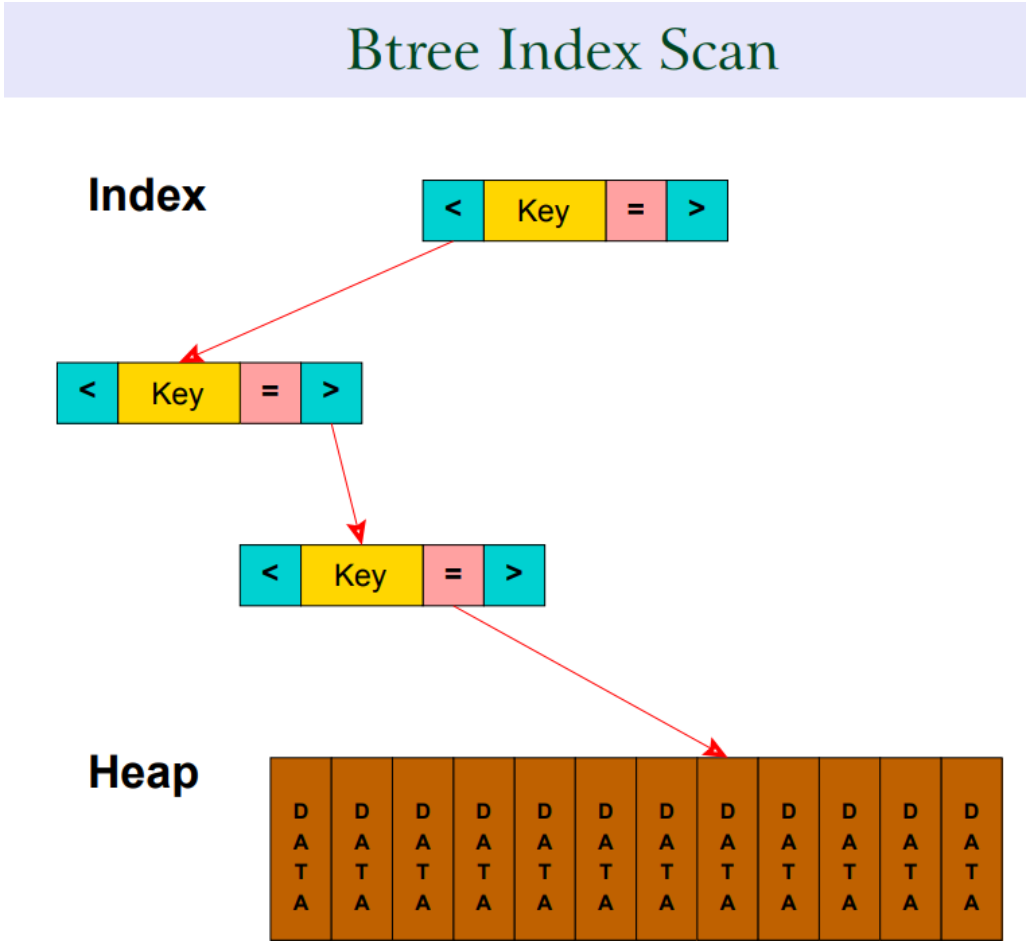
多会话并行扫描单表，优化，尽量步调一致，一个BLOCK一次IO（尽量）

分批处理，并不会把shared buffer塞满
https://github.com/digoal/blog/blob/master/201804/20180414_02.md

index only scan



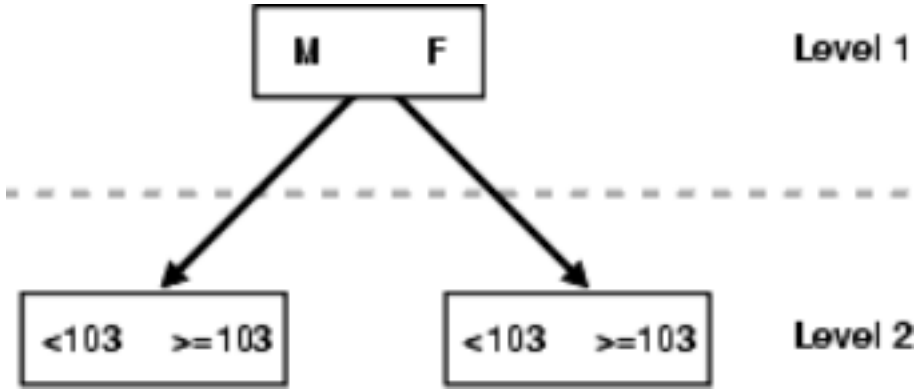
index scan


Btree Index Scan
Index
< Key = >
< Key = >
< Key = >
Heap
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA

index skip scan

- https://github.com/digoal/blog/blob/master/201803/20180323_03.md
- 从150多毫秒，降低到了0.256毫秒

```
create table t (
  sex int,
  name text
);
insert into t select random(),
md5(random()::text) from
generate_series(1,10000000); create
index idx_t on t(sex,name); select * from
t where name='abc';
```

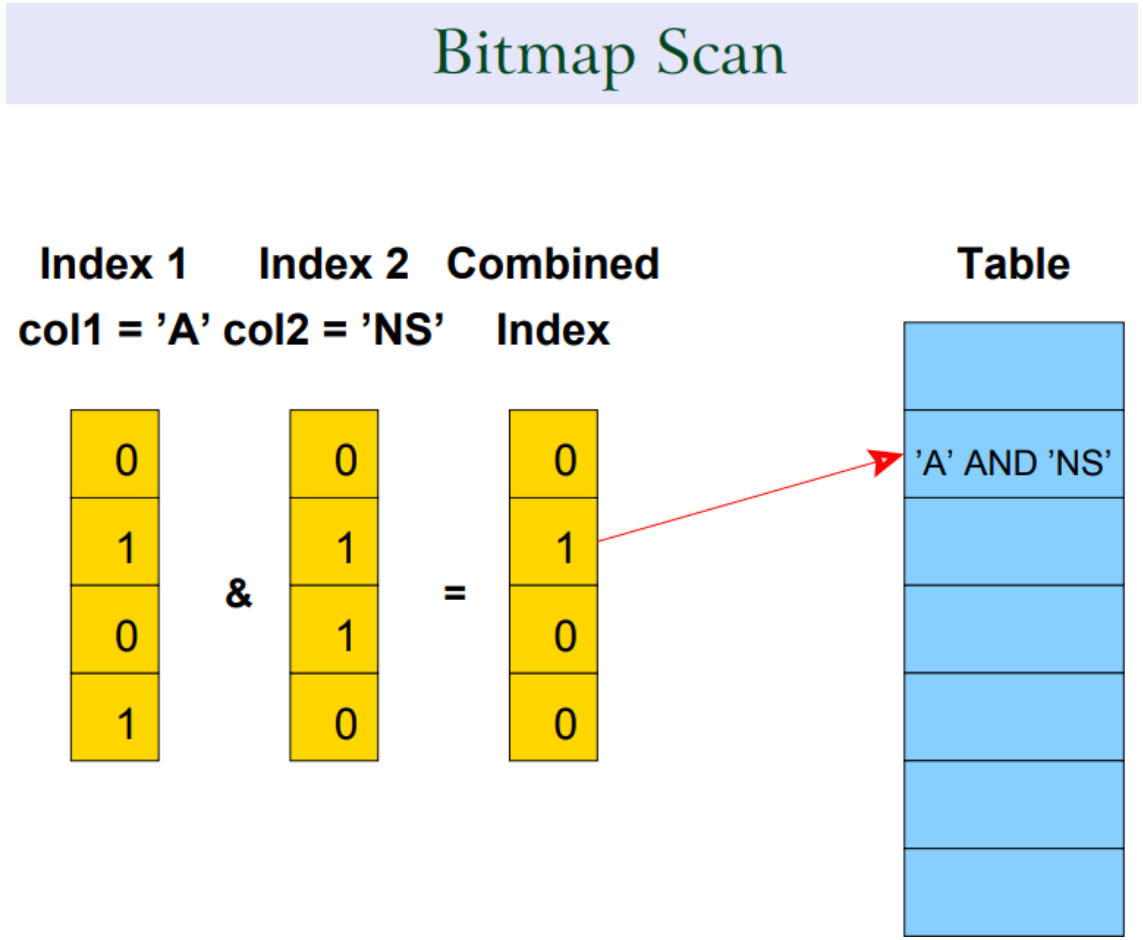


index skip scan

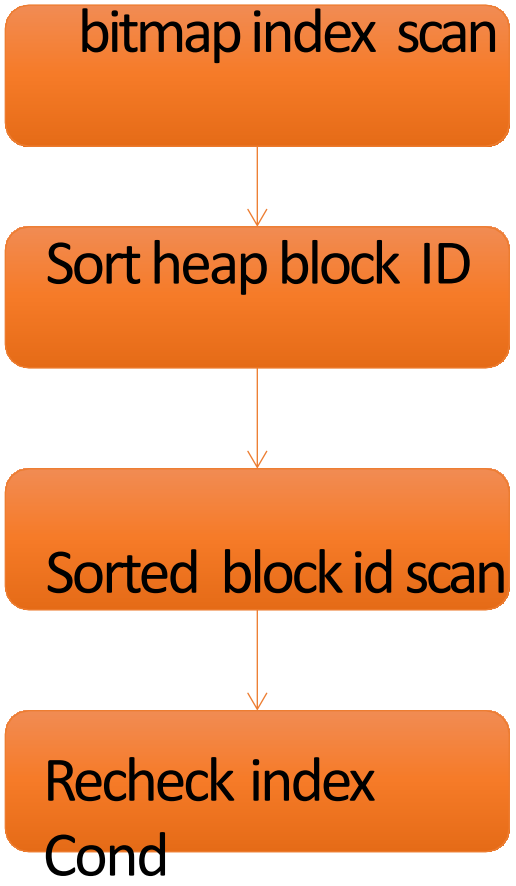
```
postgres=# explain (analyze,verbose,timing,costs,buffers) select * from t where name='abc';
                                                                 QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------
 Index Only Scan using idx_t on public.t  (cost=0.56..154297.59 rows=1 width=37) (actual time=259.064..259.064 rows=0 loops=1)
   Output: sex, name
   Index Cond: (t.name = 'abc'::text)
   Heap Fetches: 0
   Buffers: shared hit=71432
 Planning time: 0.299 ms
 Execution time: 259.092 ms
(7 rows)

Time: 259.999 ms
postgres=# explain (analyze,verbose,timing,costs,buffers) select * from t where name='abc' and sex in (0,1);
                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------
 Index Only Scan using idx_t on public.t  (cost=0.56..4.45 rows=1 width=37) (actual time=0.079..0.079 rows=0 loops=1)
   Output: sex, name
   Index Cond: ((t.sex = ANY ('{0,1}'::integer[])) AND (t.name = 'abc'::text))
   Heap Fetches: 0
   Buffers: shared hit=7 read=4
 Planning time: 0.176 ms
 Execution time: 0.099 ms
(7 rows)

Time: 0.792 ms
postgres=# explain (analyze,verbose,timing,costs,buffers) select * from t where name='abc' and sex = any (array( select * from (values (0),(1)) t (sex) ));
                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------
 Index Only Scan using idx_t on public.t  (cost=0.59..17.84 rows=1 width=37) (actual time=0.059..0.059 rows=0 loops=1)
   Output: t.sex, t.name
   Index Cond: ((t.sex = ANY ($0)) AND (t.name = 'abc'::text))
   Heap Fetches: 0
   Buffers: shared hit=8
   InitPlan 1 (returns $0)
     ->  Values Scan on "*VALUES*"  (cost=0.00..0.03 rows=2 width=4) (actual time=0.002..0.003 rows=2 loops=1)
           Output: "*VALUES*".column1
 Planning time: 0.127 ms
 Execution time: 0.090 ms
(10 rows)

Time: 0.636 ms
```

bitmap scan

1、 multi-index combine OR internal combine(GIN)



2、消除离散、重复读HEAP



bitmap scan

- index scan - > sort heap blockid - > scan heap block - > recheck index Cond.
- 优化器参考指标，相关性
- IO 放大问题消除
- SQL例子
  - https://github.com/digoal/blog/blob/master/201804/20180402_01.md

bitmap scan

```
-[ RECORD 2 ]----------+------------------
schemaname             | public
tablename              | corr_test
attname                | c2
inherited              | f
null_frac              | 0
avg_width              | 4
n_distinct             | -0.51138
most_common_vals       | {426318,766194,85
most_common_freqs      | {6.66667e-05,6.66
histogram_bounds       | {271,106225,20156
correlation            | 0.00410469  # 线性
most_common_elems      |
most_common_elem_freqs |
elem_count_histogram   |
```

bitmap scan

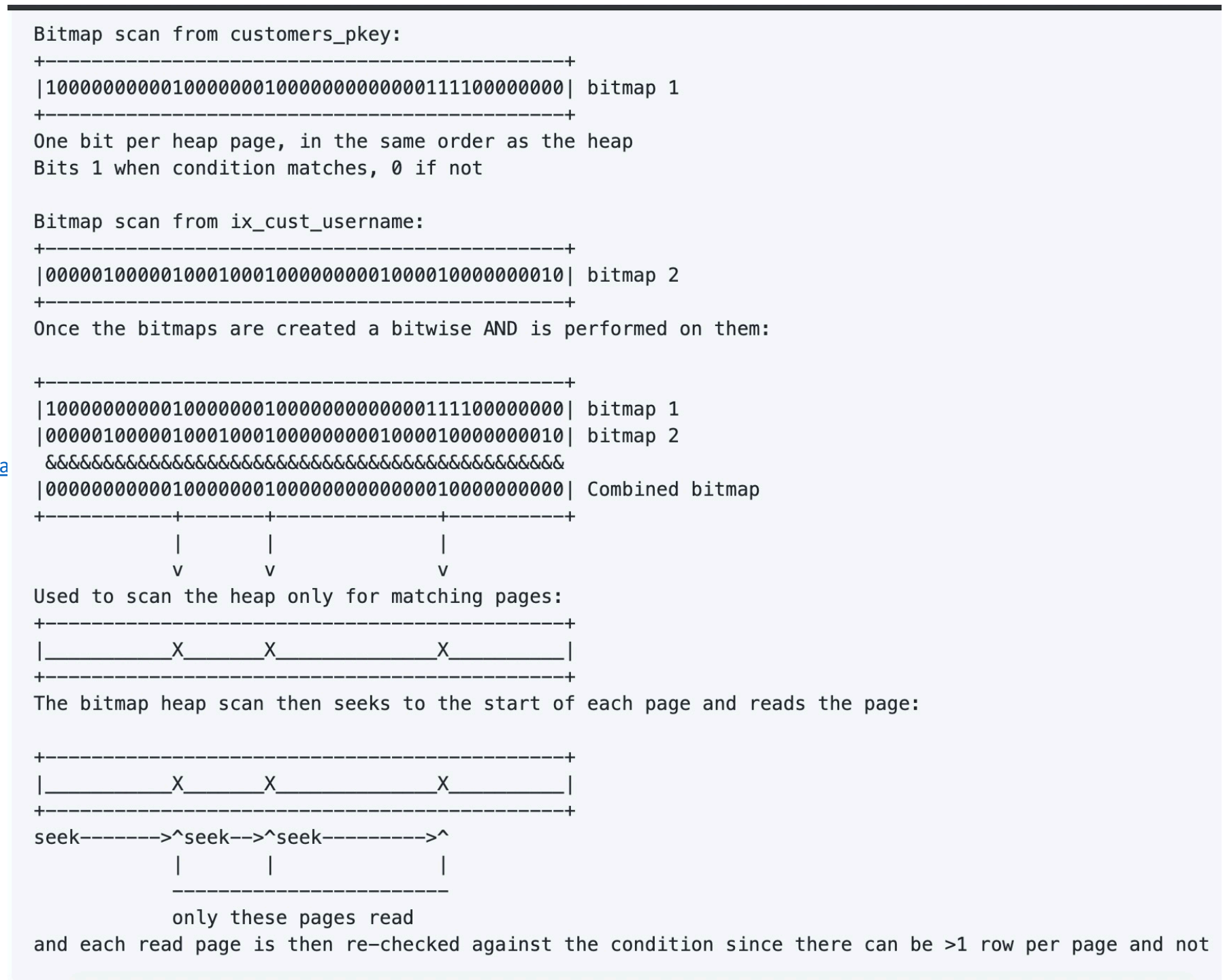
- 离散扫描，每个BLOCK几乎都被重复扫描了140次，一个BLOCK刚好存储140条记录，说明这140条记 录在顺序上完全离散。
- postgres=# explain (analyze,verbose,timing,costs,buffers) select * from corr_test where c2 between 1 and 10000000;
- QUERY PLAN
- ---
- Index Scan using idx_corr_test_2 on public.corr_test (cost=0.43..36296.14 rows=50000 width=8) (actual time=0.029..6563.525 rows=9999999 loops=1)
- Output: c1, c2
- Index Cond: ((corr_test.c2 >= 1) AND (corr_test.c2 <= 10000000))
- Buffers: shared hit=10027095
- Planning time: 0.089 ms
- Execution time: 7421.801 ms
- (6 rows)

bitmap scan

- 使用位图扫描，位图扫描的原理是从索引中得到HEAP BLOCK ID，然后按HEAP BLOCK ID 排序后顺序扫描。
- postgres=# explain (analyze,verbose,timing,costs,buffers) select * from corr_test where c2 between 1 and 10000000;
- QUERY PLAN
- ---
- Bitmap Heap Scan on public.corr_test (cost=2844700.76..3038949.24 rows=10000032 width=8) (actual time=688.150..1939.259 rows=9999998 loops=1)
- Output: c1, c2
- Recheck Cond: ((corr_test.c2 >= 1) AND (corr_test.c2 <= 10000000))
- Heap Blocks: exact=44248
- Buffers: shared hit=71573
- -> Bitmap Index Scan on idx_corr_test_2 (cost=0.00..2842200.75 rows=10000032 width=0) (actual time=681.488..681.488 rows=9999998 loops=1)
- Index Cond: ((corr_test.c2 >= 1) AND (corr_test.c2 <= 10000000))
- Buffers: shared hit=27325
- Planning time: 0.147 ms
- Execution time: 2758.621 ms
- (10 rows)

合并扫描

https://github.com/digoal/blog/blob/ma



ctid scan

- 根据给定行号，直接扫描HEAP PAGE。
- postgres=# explain (analyze,verbose,timing,costs,buffers) select * from car where ctid='(0,1)';
- QUERY PLAN
- ---

- Tid Scan on public.car (cost=0.00..1.11 rows=1 width=61) (actual time=0.006..0.007 rows=1 loops=1)
- Output: id, pos, sites, rest_sites, mod_time, order_pos
- TID Cond: (car.ctid = '(0,1)'::tid)
- Buffers: shared hit=1
- Planning time: 0.183 ms
- Execution time: 0.028 ms
- (6 rows)

ctid scan

- 根据给定行号，直接扫描HEAP PAGE。
- postgres=# explain (analyze,verbose,timing,costs,buffers) select * from car where ctid = any(array['(0,1)'::tid, '(0,2)'::tid, '(100,1)'::tid]);
- QUERY PLAN
- ---
- Tid Scan on public.car (cost=0.00..3.33 rows=3 width=61) (actual time=0.005..0.032 rows=3 loops=1)
- Output: id, pos, sites, rest_sites, mod_time, order_pos
- TID Cond: (car.ctid = ANY ('{"(0,1)","(0,2)","(100,1)"}'::tid[]))
- Buffers: shared hit=2 read=1
- Planning time: 0.182 ms
- Execution time: 0.049 ms
- (6 rows)
- 应用场景：
- Update|delete limit, parallel big update|delete
- https://github.com/digoal/blog/blob/master/201608/20160827_01.md

目录

- 任意字段组合查询
- 索引结构和原理
- 数据扫描方法
- 应用实践
  - 128个字段，任意字段组合搜索-gin,bitmapscan
  - 任意字段组合搜索-非字典化（column_prefix）gin\rum倒排搜索
  - 任意字段组合搜索-字典化gin\rum倒排搜索
  - 时空、数组、标量等多维度搜索(50倍提速vs ES)

任意字段组合查询设计1

- 每个字段都创建一个索引：

- 等值、范围、排序字段：btree
- 空间、范围字段：gist
- 多值列、全文检索、json、模糊查询字段：gin

- 查询时，数据库会自动选择bitmap index scan或者index scan

任意字段组合查询设计2

- 等值查询列：联合 rum/gin (等值、包含、相交)

- 非等值查询列选择以下索引：
- 范围、排序字段：btree
- 空间、范围字段：gist
- 多值列、全文检索、json、模糊查询字段：gin

任意字段组合查询设计3

- 组合等值、不等值查询列：prefix+value　　array 化: rum/gin

- 加字段名前缀离散化方法。

任意字段组合查询设计4

- 组合等值、不等值查询列：字典化: rum/gin
  - rum - - 目前rds pg 11支持，即将在所有常用版本支持

字典化离散化方法。

例子

- PostgreSQL 任意字段组合查询 - 含128字段，1亿记录，任意组合查询
- https://github.com/digoal/blog/blob/master/201903/20190320_02.md

例子

- 大宽表任意字段组合查询索引如何选择(btree, gin, rum)
- https://github.com/digoal/blog/blob/master/201808/20180803_01.md

例子

- ADHoc(任意字段组合)查询(rums索引加速) - 非字典化，普通、数组等组合字段生成新数组
- https://github.com/digoal/blog/blob/master/201805/20180518_02.md

例子

- ADHoc(任意字段组合)查询 与 字典化 (rum索引加速)
- https://github.com/digoal/blog/blob/master/201802/20180228_01.md

例子

- 空间应用 - 高并发空间位置更新、多属性KNN搜索并测（含空间索引）末端配送、新零售类项目
- https://github.com/digoal/blog/blob/master/201711/20171107_48.md

参考资料

- 多维组合-任意维度组合搜索案例
  - https://github.com/digoal/blog/blob/master/201903/20190320_02.md
  - https://github.com/digoal/blog/blob/master/201808/20180803_01.md
  - https://github.com/digoal/blog/blob/master/201805/20180518_02.md
  - https://github.com/digoal/blog/blob/master/201802/20180228_01.md
  - https://github.com/digoal/blog/blob/master/201711/20171107_48.md
- 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

- PG支持哪几种索引
- PG支持哪几种扫描方法
- 多值列应该使用什么索引
- 多个字段任意组合搜索应该使用什么类型的索引
- 空间包含、相交、距离排序等查询应该使用什么类型的索引
- 多个字段等值组合查询，可以使用哪几种索引加速

技术社群

PG技术交流钉钉群(3600+人)