PostgreSQL在阿里的应用

digoal

4/13/2017

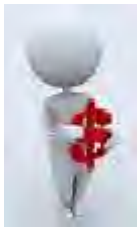
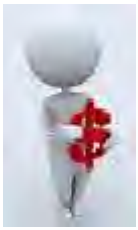
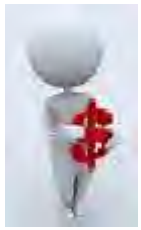
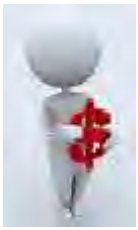
目录

- 海量导购文实时去重
- 精准广告投放
- TOB实时画像
- 任意字段组合
- 任意字段模糊匹配
- 其他 菜鸟、高德

导购业务介绍

抄袭、
重复导购，影响用户体验

导购文审核发展历程

… …

… …

导购文1

导购文n

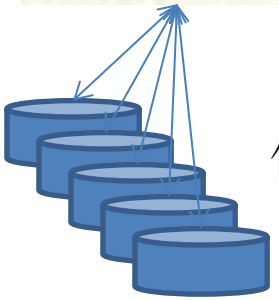
导购文1

导购文n

能耗大
效率低
延迟高

隔天
通知


小作坊



亿级文章

数据结构

导购文章ID	商品ID清单(数组)	其他字段
1	{237675,4661601,7866637,1488686,6727125,4429671,.....}	
2	{2236159,1910889,8347171,9808321,984302,.....}	
3	{1096653,3653819,9251346,6068360,8292737,603196,.....}	
...		

仿真数据

overlap = ?

导购文n

{9032400,4389590,650321,7262898,1704250,8295282,9849186,.....}

PostgreSQL GIN索引

- 数组，GIN索引，倒排效果
- postgres=# select ctid,unnest(info) from arr;
- ctid | unnest
- --------+--------
- (0,1) | 9632
- (0,1) | 6798
- (0,1) | 2069
- (0,1) | 3533
- (0,1) | 2702
- (0,1) | 3191
- (0,1) | 5561
- (0,1) | 8756
- (0,1) | 4391
- (0,1) | 3290
- (0,2) | 4179
- (0,2) | 8012
- (0,2) | 6926
- .....

倒排效果

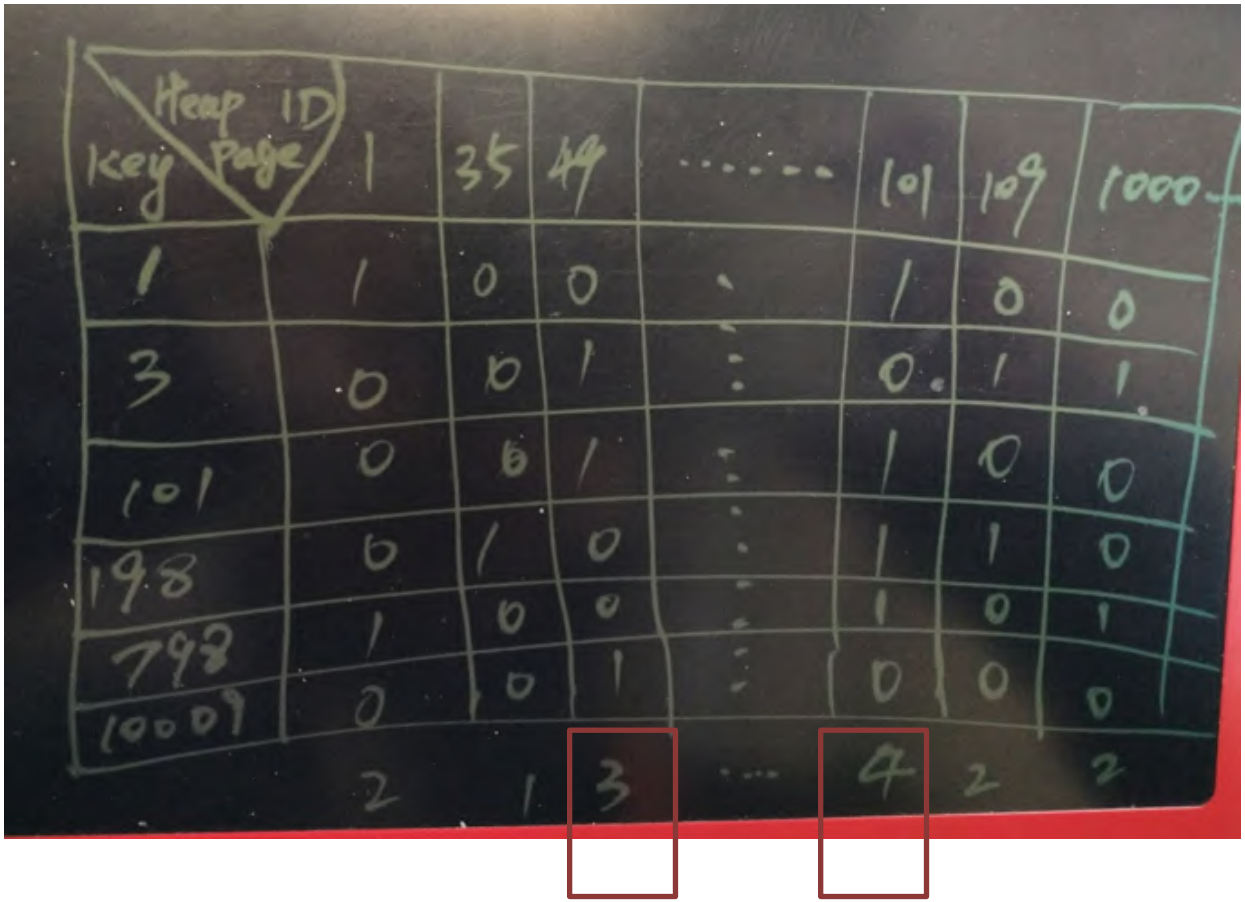
```
postgres=# select id,array_agg(ctid) from (select
unnest(info) id,ctid from arr) t group by 1 order
by 1;
..  value | 行号(s)
  1491 | {"(0,39)","(1,3)"}
  1496 | {"(0,18)"}
  1500 | {"(0,23)","(0,31)","(0,76)"}
...
```

GIN索引, OVERLAP相似检索举例

1. 新增导购文章商品ID(value), 所在数据块ID
2. 统计对应数据块value出现次数。

第一重过滤

- 收敛 BLOCK ID (if overlap>=3)
- if overlap > 4 直接返回false(不相似)



第二重过滤

- CPU CHECK BLOCK's tuples

- BLOCK: 49
- BLOCK: 101

仿真数据

沉淀导购文章	6000万+
涉及商品数量	1000万+
平均每篇导购文章涉及商品数量	11 - 50个商品
热点商品	50万热点商品
热点商品(比如iphone)被推荐次数	1000万+

单机
32Core, 128G, SSD

导购推荐平台-例子

- 测试方法，如何造数据？
- https://github.com/digoal/blog/blob/master/201701/20170112_02.md

- create extension smlar;
- create unlogged table test (id serial, -- 文章ID
- arr int8[] -- 商品ID组成的数组，假设商品ID为int8类型，那么数组就是int8[]);

- 插入5000万记录，要求如下
- int8 取值范围1~1000万， 即历史上被推荐的商品有1000万个。
- int8[] 数组长度 11 ~ 50， 即每篇导购文章，包含11到50个商品。

- 调用一次插入40条记录。
- create or replace function f() returns void as $$
- declare begin
- for i in 11..50 loop
- insert into test (arr) select array_agg((10000000*random())::int8) from generate_series(1,i);
- end loop;
- end; $$ language plpgsql strict;

生成仿真数据

- 使用pgbench调用以上函数，将生成5000万测试数据
- vi test.sql select f(); pgbench -M prepared -n -r -P 1 -f ./test.sql -c 100 -j 100 -t 12500

- 生成1000万热点商品的推荐数据
- 假设商品ID范围在 1 ~ 50万 的为热点商品，被1000万篇文章推荐过。

- create or replace function f() returns void as $$ declare begin
- for i in 11..50 loop
- insert into test (arr) select array_agg((500000*random())::int8) from generate_series(1,i);
- end loop; end; $$ language plpgsql strict;

- 使用pgbench调用以上函数，生成1000万热点数据
- pgbench -M prepared -n -r -P 1 -f ./test.sql -c 100 -j 100 -t 2500

创建GIN索引

- set maintenance_work_mem='64GB';
- create index on test using gin (arr _int8_sml_ops);

性能

响应时间测试数据

测试CASE	响应时间
普通商品40个，其中39个与历史导购文章重复	4ms
普通商品40个，其中20个与历史导购文章重复	4ms
热点商品10个,普通商品30个，其中39个与历史导购文章重复	6ms
热点商品10个,普通商品30个，其中20个与历史导购文章重复	6ms
热点商品40个，其中39个与历史导购文章重复	15ms
热点商品40个，其中20个与历史导购文章重复	15ms

吞吐量测试数据

测试CASE	TPS
普通商品35个，热点商品5个，overlap=35	9455

小结


传统方法
全表扫描匹配相似度，效率低，资源消耗巨大。
审核延迟非常高，无法实时。
PostgreSQL GIN+smlar插件。
效率高，资源消耗低。
实时判定数组相似度，(实时审核导购内容)。

目录

- 海量导购文实时去重
- 精准广告投放
- TOB实时画像
- 任意字段组合
- 任意字段模糊匹配
- 其他 菜鸟、高德

业务介绍

数据结构

用户ID	轨迹1 浏览店铺次数	轨迹2 浏览商品次数	轨迹3 购买商品数量	地理位置	其他字段
UID1	{71176739:98,26495976:82,...}	{48882486:53,58733598:56,...}	{4528019:94,66409076:62,...}	经纬度	
UID2					
...					

仿真数据

业务需求：

1. 浏览某个商品超过 ？ 次的人群。
2. 浏览某些商品超过 ？ 的人群
3. 某个区域，浏览某些商品超过 ？ 的人群

.....

体量

用户量级	百亿
商铺量级	亿
商品量级	亿
单个用户轨迹1平均量级	千
单个用户轨迹2平均量级	千
单个用户轨迹3平均量级	千

阶梯化举例

轨迹1(浏览店铺次数)
阶梯化
1 -> 0
2 -> 1-10
3 -> 11-100
4 -> 101-500
5 -> 501-
…
9 -> 10000+

轨迹2(浏览商品次数)
阶梯化
1 -> 0
2 -> 1-50
3 -> 51-200
4 -> 201-500
5 -> 501-
…
9 -> 10000+

轨迹3(购买商品数量)
阶梯化
1 -> 0
2 -> 1-10
3 -> 11-20
4 -> 21-30
5 -> 31-
…
20 -> 300+

阶梯化举例

用户ID	轨迹1 浏览店铺次数	轨迹2 浏览商品次数	轨迹3 购买商品数量	地理位置	其他字段
UID1	{71176739:98,26495976:82,...}	{48882486:53,58733598:56,...}	{4528019:94,66409076:62,...}	经纬度	
UID2					
...					

仿真数据

```
新值起始ID = new_start_val = 1                    -- 常量，用户可以自由指定，
对应维度步长(比如流量店铺的阶梯数) = step = 9     -- 常量，用户可以自由指定(
店铺ID = dp_id                                    -- 指原来的店铺ID
int/int8新值 = new_val                            -- 生成的，带有两层含义（后

已知店铺ID求new_val(写入、查询过程):

$new_val = new_start_val + (step+1)*(dp_id-1)

已知new_val求店铺ID(翻译过程):

$dp_id = 1 + (new_val-new_start_val)/(step+1)
```

用户ID	轨迹1 浏览店铺次数	轨迹2 浏览商品次数	轨迹3 购买商品数量	地理位置	其他字段
UID1	{newval1,newval2,...}	{newval1,newval2,...}	{newval1,newval2,...}	经纬度	
UID2					
...					

定向圈人

Operator	Description	Example	Result
@>	contains	ARRAY[1,4,3] @> ARRAY[3,1]	t
<@	is contained by	ARRAY[2,7] <@ ARRAY[1,7,4,2,6]	t
&&	overlap (have elements in common)	ARRAY[1,4,3] && ARRAY[2,1]	t

数据总量	分区数	标签总量	个人 平均标签数	索引	性能 (命中1.5万人群)
3.2亿	64	400万	4000	GIN	1毫秒

- 性能
- GIN
- 分区(按用户ID)
  - 单机多核并行append
- sharding(按用户ID)
  - 多机并行

目录

- 海量导购文实时去重
- 精准广告投放
- TOB实时画像
- 任意字段组合
- 任意字段模糊匹配
- 其他 菜鸟、高德

业务背景

- ToB的用户推荐系统
- 业务数据包括USERID，APPID，TAGs。
  - 根据APPID+TAG组合，搜索复合条件的USERID
  - 查询单个用户的TAG
- 数据总量约10亿
- 单个APPID(TOB)用户数设计量级1亿
- TAG设计量级1万
- 规模：10万亿user_tags

贴标签 - 业务指标

- **分钟级延迟**
  - 写入标签数据
  - 删除用户标签
  - 更新用户标签

圈人 - 业务指标

- 并发指标
  - 约**200~300**个ToB用户根据对应的APPID，根据TAG组合筛选需要的用户群体。
- 查询需求: 按TAG组合
  - 包含，不包含，或，与
  - and, or, not
- 响应时间指标
  - **毫秒级**

开脑洞

- 表结构设计
  - appid, userid, tag1, tag2, ..., tag10000
- 查询SQL写法
  - select appid,userid from tbl where tag1 **and** tag2 **and** ... **or** (not tagn) **or** (tagx **and** tagxx **or** tagxxx) ......
- 问题
  - 全字段索引？
  - 超宽表？
  - 存储空间？
  - 插入效率、查询效率？

用户的老方案

- 目前没有产品可以支撑**1万列**的宽表
- 拆成N张表(每张表1000个字段)，通过PK关联
- 目前规模1亿，使用了8台主机（ADS）
- TAG更新延迟1天
- 查询响应 - 分钟级

方案1

- 数据结构
  - APPID， USERID， **TAG[]数组**
  - 单个数组最大长度1GB(约支持2.6亿个TAG(int4))
- 按APPID分区，随机分片
- query语法
  - 包含array2指定的所有TAG
    - 数组1包含数组2的所有元素
    - array1 @> array2
    - **支持索引检索**
  - 包含array2指定的TAG之一
    - 数组1与数组2有重叠元素
    - array1 && array2
    - **支持索引检索**
  - 不包含array2指定的所有tag
    - 数组1与数组2没有重叠元素
    - not array1 && array2
    - **不支持索引检索**

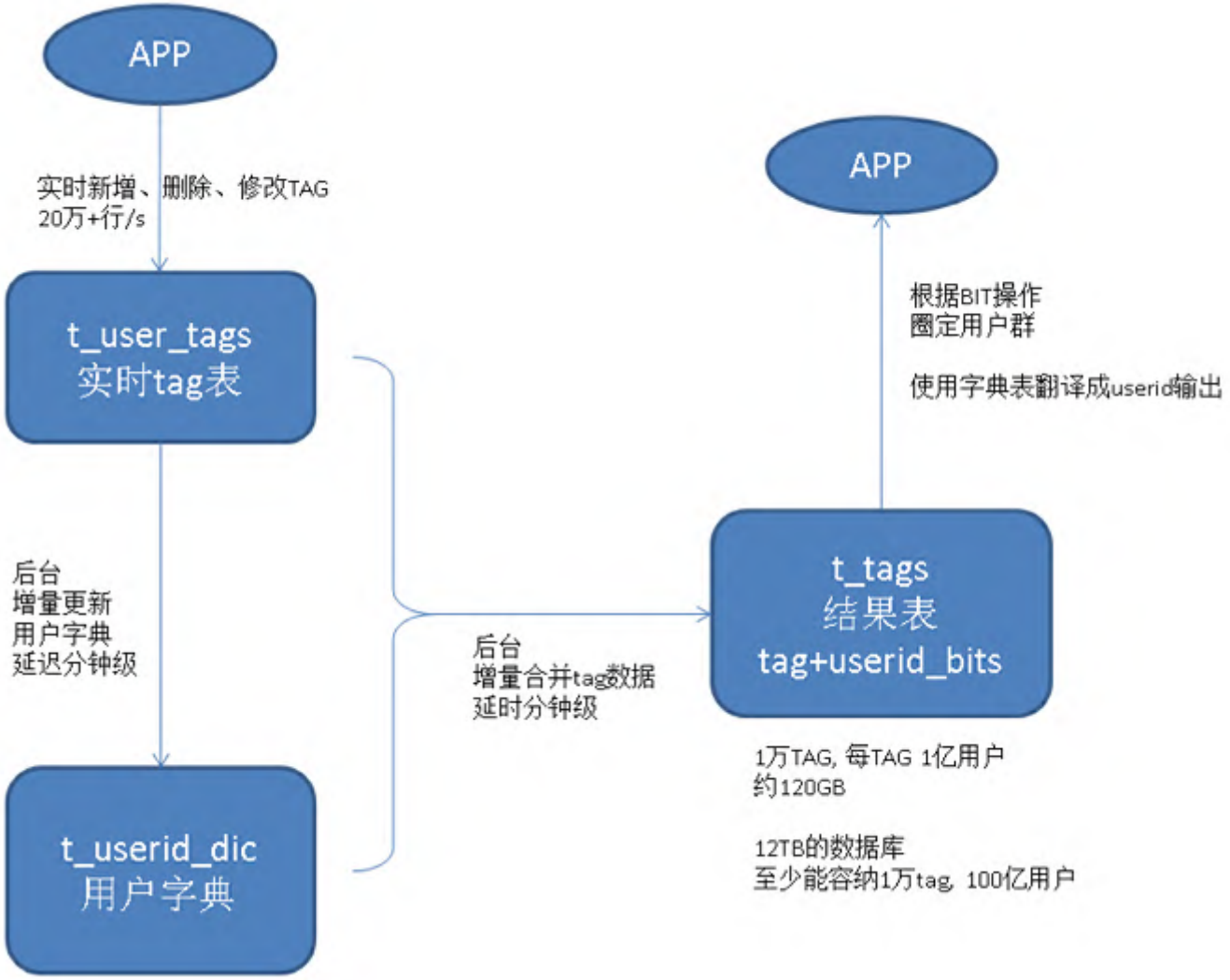
方案2

- 数据结构
  - APPID， USERID， TAG比特流
  - 单个BIT字段最大支持1GB长度BIT流（支持85亿个TAG）
  - 每个BIT代表一个TAG
- 按APPID分区，随机分片
- query语法
  - **都不支持索引，但是可以使用CPU多核并行**
  - 包含bit2指定的所有TAG(需要包含的TAG对应的BIT设置为1，其他为0)
    - bitand(tag_bit,bit2) = bit2
  - 包含bit2指定的TAG之一(需要包含的TAG对应的BIT设置为1，其他为0)
    - bitand(tag_bit,bit2) > 0
  - 不包含bit2指定的所有tag (需要包含的TAG对应的BIT设置为1，其他为0)
    - bitand(tag_bit,bit2) = zerobit(10000) -- '0000...0000000'

方案3

- 结构设计
  - appid, tagid, userid比特流(1亿用户12MB)
- 延迟合并 + query时合并
  - data -> 明细表 -> 增量聚合 -> appid,tagid,userid
- query
  - 都支持索引
  - 包含这些tags的用户
    - userids (bitand) userids .....
    - 结果为bit位为1的用户
  - 不包含这些tags的用户
    - userids (bitor) userids .....
    - 结果为bit位为0的用户
  - 包含这些tags之一的用户
    - userids (bitor) userids .....
    - 结果为bit位为1的用户

方案3 - 增量合并


APP
实时新增、删除、修改TAG
20万+行/s
t_user_tags
实时tag表
后台
增量更新
用户字典
延迟分钟级
t_userid_dic
用户字典
后台
增量合并tag数据
延时分钟级
t_tags
结果表
tag+userid_bits
1万TAG, 每TAG 1亿用户
约120GB
12TB的数据库
至少能容纳1万tag, 100亿用户
APP
根据BIT操作
圈定用户群
使用字典表翻译成userid输出

TAG类型 数组 vs bit

- 空间占用
  - 数组，与用户平均TAG数有关
    - **1万个元素约80KB**
      - select pg_column_size(id) from (select (select array_agg(10000*random()) from generate_series(1,10000)) id from generate_series(1,1)) t;
      - 80024
  - BIT，固定长度
    - **1万个bit约1.25K**
      - select pg_column_size(id) from (select (select (string_agg(mod((2*random())::int,2)::text,''))::varbit from generate_series(1,10000)) id from generate_series(1,1)) t;
      - 1258
- 索引支持
  - 数组支持(包含，包含任意)的索引查询，不支持(不包含)的索引查询
  - bit不支持本CASE的索引（使用方案3无需bit索引）
- 性能

空间评估

- 1亿用户，1万标签
- 方案1
  - 裸数据约8TB
  - 索引约8TB
- 方案2
  - 裸数据约120GB
  - 无索引，全量计算
- 方案3
  - 裸数据约120GB
  - 索引很小，MB级(仅仅1万条记录)

方案1 - 性能指标

- 方案1
  - 如果使用方案1，建议按USERID拆库(使用plproxy并行)
- 性能
  - 插入 速度(10w/s)
  - 更新 tag速度(10w/s)
  - 删除 tag速度(10w/s)
  - 查询速度(返回结果集10万条,1秒左右)
  - 并发指标(500+)

方案2 - 性能指标

- 方案2
- BIT
  - 插入速度
    - **(批量)每秒插入24.5万, 326MB/s**
  - 更新、删除 tag速度
    - create or replace function randbit(int) returns varbit as $$
    - select (string_agg(mod((2*random())::int,2)::text,''))::varbit from generate_series(1,$1);
    - $$ language sql strict volatile;
    - create or replace function zerobit(int) returns varbit as $$
    - select (string_agg('0',''))::varbit from generate_series(1,$1);
    - $$ language sql strict immutable;
    - update t_bit set tags=randbit(10000) where userid=:id;
    - **每秒更新、删除1万 记录，响应时间约4毫秒**
  - 查询速度
    - do language plpgsql $$
    - declare
    - sql text;
    - bit1 varbit := randbit(10000);
    - bit2 varbit := randbit(10000);
    - bit3 varbit := randbit(10000);
    - zbit varbit := zerobit(10000);
    - begin
    - set max_parallel_workers_per_gather =27;
    - sql := 'select * from t_bit where bitand(tags,'''||bit1::text||''')='''||bit1::text||''' and bitand(tags,'''||bit2::text||''')>bit''0'' and bitand(tags,'''||bit3::text||''')='''||zbit::text||'''';
    - raise notice '%', sql;
    - -- execute sql;
    - end;
    - $$;
    - **开27个并行, 17秒。**
  - 并发指标

方案3 - 性能指标

- 方案3
- BIT
  - 前端写入速度
    - **每秒插入24.5万, 326MB/s，满足实时标签修正**
  - 合并速度
    - (批量)10亿用户bit/秒, 100TAG/s。满足分钟级延迟。
  - 更新、删除 tag速度
    - 每个TAG每次变更1万个用户。
    - 100 tag/s, 100毫秒/tag，支持tag并行更新。
    - 换算成user tag绑定/删除 ops ~= 100万 users/s。
  - 查询速度
    - 任意条件组合
    - **毫秒级**返回用户数据
    - 支持流式返回用户集合
  - 并发指标
    - 使用游标返回数据
    - **支持500+并发**

前后对比

	主机数	用户数\标签量	更新延迟	查询并发	查询响应
优化前	8台物理机	1亿用户 1万TAG	天	200	分钟
优化后	**1台RDS PG**	**100亿用户 1万TAG**	**分钟**	**500+**	**毫秒**

方案3 - 后期优化

- 多行优化（同一个tag占多行，每行存储部分USERID BITS），减少垃圾版本大小(改少量BIT)，减少锁冲突(不同分段可以同时更新,行锁原因)
  - **roaring bitmap数据类型**
  - **https://github.com/zeromax007/gpdb-roaringbitmap**
  - **https://github.com/RoaringBitmap/CRoaring**
- BIT字段块级元数据(比如每个块有多少0,1？)

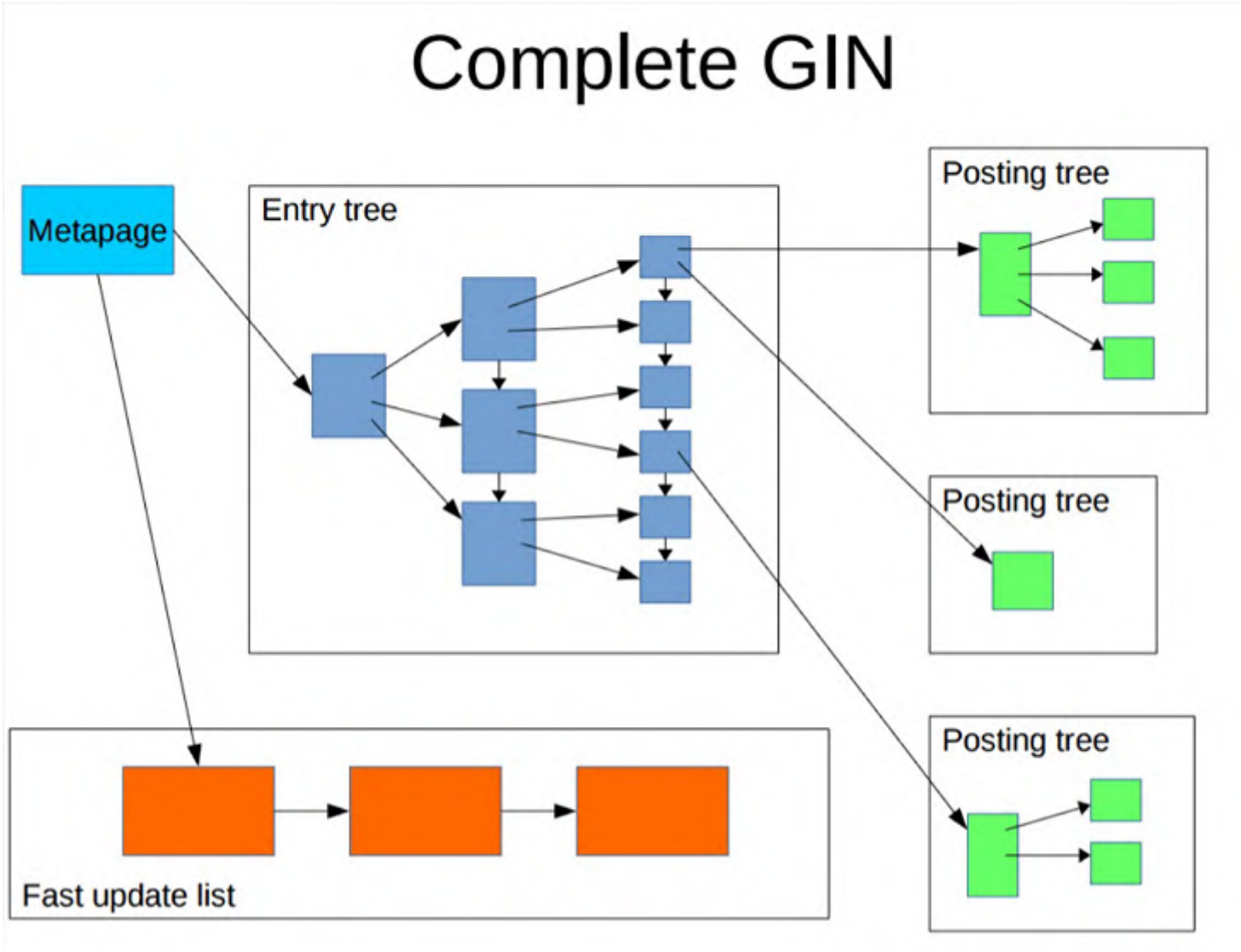
目录

- 海量导购文实时去重
- 精准广告投放
- TOB实时画像
- 任意字段组合
- 任意字段模糊匹配
- 其他 菜鸟、高德

业务介绍

该在哪些字段建索引??????

GIN组合索引 - 任意组合查询



指标

记录数	字段数	表大小	全字段索引大小
1000万	6	1.2GB	3.7GB

返回记录条数	任意1列	任意2列 or	任意3列 or	任意4列 or	任意5列 or	任意6列 or
1	0.15毫秒	2.9毫秒	2.9毫秒	2.9毫秒	2.9毫秒	2.9毫秒
10	1.5毫秒	2.9毫秒	2.9毫秒	2.9毫秒	2.9毫秒	2.9毫秒
1000	3毫秒	4.3毫秒	4.3毫秒	4.3毫秒	4.3毫秒	4.3毫秒

返回记录条数	任意1列	任意2列 and	任意3列 and	任意4列 and	任意5列 and	任意6列 and
1	0.15毫秒	0.3毫秒	0.3毫秒	0.3毫秒	0.3毫秒	0.3毫秒

目录

- 海量导购文实时去重
- 精准广告投放
- TOB实时画像
- 任意字段组合
- 任意字段模糊匹配
- 其他 菜鸟、高德

业务介绍

全文检索？？
www.xxxxxx.com
xxxx公司

搜索引擎？？
同步、一致性、
额外费用

模糊查询
全表、全字段
支持中文

pg_trgm

- gin\rum索引
- 支持正则表达式、前后模糊索引检索
- 亿级数据、毫秒级响应

目录

- 海量导购文实时去重
- 精准广告投放
- 实时画像
- 任意字段组合
- 任意字段模糊匹配
- 敬请期待 - IoT、菜鸟、高德、B2B、阿里云、安全部、 ……
  - GIS、流计算、HTAP、证券、复杂运算、……

谢谢

- GIT
- https://github.com/digoal/blog/blob/master/README.md

高效运维社区
GreatOPS Community

会议

- 3月18日 DevOpsDays 北京
- 8月18日 DevOpsDays 上海
- 全年 DevOps China 巡回沙龙
- 4月21日 GOPS深圳
- 11月17日 DevOps金融上海

培训

- EXIN DevOps Master 认证培训
- DevOps 企业内训
- DevOps 公开课
- 互联网运维培训

咨询

- 企业DevOps 实践咨询
- 企业运维咨询

商务经理：刘静女士
电话 / 微信：13021082989
邮箱：liujing@greatops.com

Thanks

高效运维社区

开放运维联盟

荣誉出品

想第一时间看到
高效运维社区公众号
的好文章吗？

请打开高效运维社区公众号，点击右上角小人，如右侧所示设置就好