PostgreSQL 慢 SQL 调优手册

目录

1、Create Index Directly

```
优化前
explain analyse
select id from test where update_flag = true;
                                        QUERY PLAN
--------------------------------------------------------------------------------------------------
 Seq Scan on test  (cost=0.00..6711.12 rows=12 width=338) (actual time=0.439..109.827 rows=18 loops=1)
   Filter: update_flag
 Total runtime: 109.941 ms

查看表test结构，字段update_flag有default值false，整个表中update_flag字段为true较少，
且发现字段update_flag上面没有建index，所以建index可以提升查询速度，建index后

优化后
explain analyze
select id from test where update_flag = true;
                                        QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------
 Index Scan using test_update_flag_idx on test  (cost=0.00..105.57 rows=473 width=338)(actual time=0.023..0.049 rows=11 loops=1)
   Index Cond: (update_flag = true)
   Filter: update_flag
 Total runtime: 0.071 ms
```

总结: 需要注意数据选择比的问题, 如果 where 条件时 update_flag=false, 效果就不会有这么明显, 因为 update_flag 字段值几乎都是 false， 尽管有 index， 其效果和 Seq Scan 时间没什么差别。

2、Change Conditions to Use Index

```
explain analyze
select
        o.id,
        to_timestamp(o.pay_time),
        o.state,
        o.money
from
        public.order o
where
            to_timestamp(pay_time)>'2012-07-26'
        and to_timestamp(pay_time)<'2012-08-02'
;

                                        QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------
 Seq Scan on "order" o  (cost=0.00..419429.47 rows=54513 width=25) (actual time=212.383..62639.878 rows=18042 loops=1)
   Filter: (
             (to_timestamp((pay_time)::double precision) > '2012-07-26 00:00:00+08'::timestamp with time zone)
         AND (to_timestamp((pay_time)::double precision) < '2012-08-02 00:00:00+08'::timestamp with time zone)
           )
   Rows Removed by Filter: 10884634
 Planning time: 0.133 ms
 Execution time: 62641.395 ms
(5 rows)

public.order的 pay_time字段如下:

 pay_time                  | bigint                    | not null default 0

"order_pay_time_idx" btree (pay_time)
```

在 pay_time 上有 index，并且是 bigint 类型，但是使用函数 to_timestamp(pay_time)转换为 timestamptz 类型后，就用不上 index 了，修改为如下：

```
explain analyze
select
      o.id,
      to_timestamp(o.pay_time),
      o.state,
      o.money
from
      public.order o
where
          pay_time > extract( epoch from '2012-07-26'::timestamptz )::bigint
      and pay_time < extract( epoch from '2012-08-02'::timestamptz )::bigint;
                                                          QUERY PLAN

------------------------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on "order" o  (cost=255.44..51000.71 rows=17853 width=25) (actual time=2.358..75.164 rows=18042 loops=1)
   Recheck Cond: ((pay_time > (date_part('epoch'::text, '2012-07-26 00:00:00+08'::timestamp with time zone))::bigint)
              AND (pay_time < (date_part('epoch'::text, '2012-08-02 00:00:00+08'::timestamp with time zone))::bigint))
   Heap Blocks: exact=2244
   ->  Bitmap Index Scan on order_pay_time_idx  (cost=0.00..250.98 rows=17853 width=0) (actual time=2.010..2.010 rows=18042 loops=1)
         Index Cond: ((pay_time > (date_part('epoch'::text, '2012-07-26 00:00:00+08'::timestamp with time zone))::bigint)
                  AND (pay_time < (date_part('epoch'::text, '2012-08-02 00:00:00+08'::timestamp with time zone))::bigint))
 Planning time: 0.221 ms
 Execution time: 76.264 ms
(7 rows)

注意 extract(epoch from '2012-07-26'::timestamptz) 即 date_part( text, timestamp with time zone) 返回类型为double precision,
    而pay_time字段类型为bigint, 所以需要进行强转

                              List of functions
   Schema   |   Name    | Result data type |        Argument data types         |  Type
------------+-----------+------------------+------------------------------------+--------
 pg_catalog | date_part | double precision | text, timestamp with time zone     | normal
```

新纪元时间 Epoch 是以 1970-01-01 00:00:00 UTC 为标准的时间，将目标时间与 1970-01-01 00:00:00 时间的差值以秒来计算 ，单位是秒，可以是负值；有些应用会将时间存储成 epoch 时间形式，以提高读取效率，下面演示下 pg 中 epoch 时间的使用换算方法。

select extract(epoch from timestamp without time zone '1970-01-01 01:00:00');

【将 time stamp 时间转换成 epoch 时间】

	date_part
▶	3600

select extract(epoch from timestamp '1970-01-01 01:00:00');

【将 time stamp 时间转换成 epoch 时间】

	date_part
▶	3600

select extract(epoch from '1970-01-01 01:00:00'::timestamp)::bigint

	date_part
▶	3600

select extract(epoch from '1970-01-01 01:00:00'::timestamp without time zone)::bigint;

	date_part
▶	3600

3、尽量避免在where子句中对字段进行运算，导致查询规划器放弃使用index

```
优化前
explain analyze select modify_count from test_account_trace where modify_count /2= 50;
                                       QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Seq Scan on test_account_trace  (cost=0.00..1764.88 rows=281 width=8) (actual time=0.015..20.857 rows=8436 loops=1)
   Filter: ((modify_count / 2) = 50)
 Total runtime: 21.777 ms
(3 rows)


优化后
explain analyze select modify_count from test_account_trace where modify_count= 50*2;
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on test_account_trace  (cost=162.05..1189.16 rows=8489 width=8)
                                         (actual time=1.702..5.711 rows=8436 loops=1)
   Recheck Cond: (modify_count = 100)
   ->  Bitmap Index Scan on test_account_trace_modify_count_idx  (cost=0.00..159.92 rows=8489 width=0)
                                                                 (actual time=1.500..1.500 rows=8436 loops=1)
         Index Cond: (modify_count = 100)
 Total runtime: 6.670 ms
(5 rows)
```

4、尽量避免在 where 子句中对字段类型进行强制转换，导致查询规划器放弃使用 index

```
优化前
explain analyze select count(1) from test where operationtime::timestamp  > now() - '2 hour'::interval;
                                          QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Aggregate  (cost=1189539.44..1189539.45 rows=1 width=0) (actual time=20900.323..20900.323 rows=1 loops=1)
   ->  Seq Scan on test  (cost=0.00..1184988.14 rows=1820519 width=0)
                         (actual time=16.660..20868.871 rows=333001 loops=1)
         Filter: ((operationtime)::timestamp without time zone > (now() - '02:00:00'::interval))
         Rows Removed by Filter: 4602334
 Planning time: 0.119 ms
 Execution time: 20900.377 ms
(6 rows)

table 结构
 operationtime              | timestamp with time zone | not null
Indexes:
    "test_operationtime_idx" btree (operationtime)

优化后
explain analyze  select count(1) from test where operationtime > now() - '2 hour'::interval;
                                       QUERY PLAN
-----------------------------------------------------------------------------------------------------------

 Aggregate  (cost=514579.06..514579.07 rows=1 width=0) (actual time=336.074..336.074 rows=1 loops=1)
   ->  Index Only Scan using test_operationtime_idx on test  (cost=0.44..513872.38 rows=282674 width=0)
                                                             (actualtime=0.030..309.101 rows=333429 loops=1)
         Index Cond: (operationtime > (now() - '02:00:00'::interval))
         Heap Fetches: 333681
 Planning time: 0.139 ms
 Execution time: 336.102 ms
(6 rows)
```

优化前

```
explain analyze
select qunar_id from test where day = 5 and type <> 4 and status = 1 and stock  > 0.0;

Total runtime: 15436.666 ms
```

由于table的定义中 stock 的数据类型是int,

```
 stock                    | integer                          |
    "test_qunar_id_idx" btree (qunar_id) WHERE day = 5 AND stock > 0 AND type <> 4 AND status = 1
```

其中, 原query中的stock > 0.0 实际上等同于 stock::numeric > 0.0，就用不到已经存在的Partial Index,

如果改变写法

```
explain analyze
select qunar_id from test where day = 5 and type <> 4 and status = 1 and stock  > 0;

                                                     QUERY PLAN
-----------------------------------------------------------------------------------------------
->  Bitmap Index Scan on test_qunar_id_idx  (cost=0.00..1.26 rows=77 width=0)
                                            (actual time=0.009..0.009 rows=6 loops=3435)
          Index Cond: (qunar_id)::text

 Planning time: 2.453 ms
 Execution time: 134.378 ms
```

5、少用 outer join，减少不必要的 sub-query 层级数【在不影响得到正确结果的前提下】

```sql
SELECT
        *
FROM (
      SELECT
                ps.*
      FROM      product ps
      LEFT JOIN supplier si --供应商表
              ON ps.supplier_id = si.id
      LEFT JOIN
                (
                SELECT
                        *
                FROM
                        (
                        SELECT
                                product_id,
                                last(reason) as reason,
                                last(operate_time) as operate_time
                        FROM b2c_parent_onoff_trace --产品操作流水表
                        WHERE product_id in
                                (SELECT id FROM   product
                                 WHERE  status in ('a','b','c') AND supplier_id in ( $1, $2, $3))
                        ) bpot
                        group by product_id
                        order by product_id
                ) as pot
              ON ps.id = pot.product_id
     ) a
WHERE status in ('a','b','c')
  AND si.shopname ~ '北京'
  AND supplier_id in ( $1, $2, $3 )
ORDER BY create_time desc,id desc;
```

由于每个产品都有其提供的供应商，所以LEFT JOIN supplier，可替换为JOIN

在 product 表里面添加了2列，

```
 last_modify_reason                | character varying(200)       | 最后操作原因
 last_modify_time                  | timestamp with time zone     | 最后操作时间
```

去掉了原query中的最后一个 LEFT JOIN 的 Sub-query

优化后的Query

```
SELECT
       ps.*
FROM   product ps
JOIN   supplier si
  ON   ps.supplier_id = si.id
  AND  status in ('a','b','c')
  AND  si.shopname ~ '北京'
  AND  supplier_id in ( $1, $2, $3 )
ORDER BY
       create_time desc,id desc;
```

6、坚决避免 select * 和 redundant columns【多余字段】

```
EXPLAIN (ANALYZE , VERBOSE, COSTS, BUFFERS, TIMING)
select
    *
from
    ( select * from a where group_id in (666,888) ) t
join
    ( select * from b where tag in ( '机票', '酒店' ) tg on t.id = tg.team_id
join
    ( select * from c where service = 'abc' ) o on t.id = o.team_id;
```

更改后

```
EXPLAIN (ANALYZE , VERBOSE, COSTS, BUFFERS, TIMING)
select
        a.id
from
        a
join
        b
  on
        a.id = b.team_id
join
        c
  on
        a.id = c.team_id
where
        a.group_id in (666,888)
    and b.tag in ( '机票', '酒店')
    and c.service = 'abc';
```

7、Index on Expressions

优化前

```
explain analyze
select
        a.arrive
from
       a
join
       b
  on
       b.id = a.route_id
where
       a.status in (1,3)
       and a.departure = '杭州'
       and a.arrive ~ E'(0x03|^)晋吉岛($|0x03)'
limit 10 ;
```

```
                                   QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=5.00..908829.64 rows=1 width=204) (actual time=3506.471..3506.471 rows=0 loops=1)
   ->  Nested Loop  (cost=5.00..908829.64 rows=1 width=204) (actual time=3506.471..3506.471 rows=0 loops=1)
         ->  Seq Scan on a  (cost=0.00..908802.57 rows=3 width=178) (actual time=3506.468..3506.468 rows=0 loops=1)
               Filter: ((departure ~ '杭州'::text) AND (arrive ~ '(0x03|^)晋吉岛($|0x03)'::text))
               Rows Removed by Filter: 1438618
         ->  Bitmap Heap Scan on route  (cost=5.00..9.01 rows=1 width=30) (never executed)
               Recheck Cond: (id = a.route_id)
               Filter: (status = ANY ('{1,3}'::integer[]))
               ->  Bitmap Index Scan on b_pkey  (cost=0.00..5.00 rows=1 width=0) (never executed)
                     Index Cond: (id = a.route_id)
 Total runtime: 3506.553 ms
```

创建index

```
CREATE INDEX CONCURRENTLY ON a USING gin (string_to_array(arrive, '\x03'::text))
```

优化后

```
explain analyze
select
        a.arrive
from
        a
join
        b
  on
        b.id = a.route_id
where
        a.status in (1,3)
        and a.departure = '杭州'
        and string_to_array(a.arrive,E'\x03') && ARRAY['普吉岛'];
                                        QUERY PLAN
------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=12592.88..26490.24 rows=108 width=204) (actual time=165.167..185.675 rows=183 loops=1)
   ->  Bitmap Heap Scan on a  (cost=12587.88..25515.86 rows=108 width=178)
                              (actual time=165.129..181.114 rows=183 loops=1)
         Recheck Cond: ((string_to_array(arrive, '\x03'::text) && '{普吉岛}'::text[]) AND (status = ANY ('{1,3}'::integer[])))
         Filter: (departure = '杭州'::text)
         Rows Removed by Filter: 3891
         ->  BitmapAnd  (cost=12587.88..12587.88 rows=3377 width=0) (actual time=160.886..160.886 rows=0 loops=1)
               ->  Bitmap Index Scan on a_string_to_array_idx  (cost=0.00..2502.64 rows=30752 width=0)
                                                               (actual time=27.548..27.548 rows=47266 loops=1)
                     Index Cond: (string_to_array(arrive, '\x03'::text) && '{普吉岛}'::text[])
               ->  Bitmap Index Scan on a_status_idx  (cost=0.00..10084.94 rows=162079 width=0)
                                                      (actual time=125.018..125.018 rows=759348 loops=1)
                     Index Cond: (status = ANY ('{1,3}'::integer[]))
   ->  Bitmap Heap Scan on route  (cost=5.00..9.01 rows=1 width=30) (actual time=0.021..0.023 rows=1 loops=183)
         Recheck Cond: (id = a.route_id)
         ->  Bitmap Index Scan on b_pkey  (cost=0.00..5.00 rows=1 width=0) (actual time=0.006..0.006 rows=20 loops=183)
               Index Cond: (id = a.route_id)
 Total runtime: 185.776 ms
```

```
explain analyze select * from test_cash where lower(cash_code::text) = lower('Qunar_Test_Code');
                                        QUERY PLAN
------------------------------------------------------------------------------------------------------------
 Seq Scan on test_cash  (cost=0.00..1781451.49 rows=80063 width=6314) (actual time=14580.250..14580.277 rows=1 loops=1)
   Filter: (lower((cash_code)::text) = 'qunar_test_code'::text)
   Rows Removed by Filter: 16012648
 Planning time: 0.141 ms
 Execution time: 14580.297 ms
```

test_cash 定义

```
 cash_code           | character varying(128)    | not null
Indexes:
    "test_cash_code_idx" btree (cash_code)

create unique index CONCURRENTLY on test_cash (lower(cash_code));

explain analyze select * from test_cash where lower(cash_code::text) = lower('Qunar_Test_Code');
                                        QUERY PLAN
------------------------------------------------------------------------------------------------------------
 Index Scan using test_cash_lower_idx on test  (cost=0.44..4.54 rows=1 width=4266) (actual time=0.019..0.019 rows=1 loops=1)
   Index Cond: (lower((cash_code)::text) = 'qunar_test_code'::text)
 Planning time: 0.285 ms
 Execution time: 0.039 ms
```

8、Partial Indexes

```
优化前
explain analyze
SELECT id,number,is_number_encrypt FROM test WHERE code=2 AND is_number_encrypt = 'f' AND number != '' order by id asc limit 5000;
                                                              QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
Limit  (cost=0.43..56605.46 rows=5000 width=16) (actual time=10338.630..10338.630 rows=0 loops=1)
->  Index Scan using test_pkey on test  (cost=0.43..1655538.85 rows=146236 width=16) (actual time=10338.628..10338.628 rows=0 loops=1)
      Filter: ((NOT is_number_encrypt) AND ((number)::text <> ''::text) AND (code = 2))
        Rows Removed by Filter: 11837573
Planning time: 0.288 ms
Execution time: 10338.648 ms

调查数据分布
select count(1) from test ;
  count
---------
 1438924

select count(1) from test where number <> '';
  count
---------
 1438855

select code, count(1) from test group by code;
 code |  count
------+---------
    0 | 1432279
    1 |    4878
    2 |     618
    3 |    1149

select is_number_encrypt, count(1) from test group by is_number_encrypt ;
is_number_encrypt |  count
------------------+----------
f                 |   162915
t                 | 11675820


create unique index CONCURRENTLY on test(id) where is_number_encrypt = 'false' and code=2;

这是一个unique Partial Index

优化后
explain analyze
SELECT id,number,is_number_encrypt FROM test WHERE code=2 AND is_number_encrypt = 'f' AND number != '' order by id asc limit 5000;
                                                              QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=0.12..4.46 rows=3 width=16) (actual time=0.003..0.003 rows=0 loops=1)
   ->  Index Scan using test_id_idx on test  (cost=0.12..4.46 rows=3 width=16) (actual time=0.002..0.002 rows=0 loops=1)
         Filter: ((number)::text <> ''::text)
 Planning time: 0.201 ms
 Execution time: 0.018 ms
(5 rows)


mydb=# \d+ test_id_idx
  Index "public.test_id_idx"
 Column |  Type   | Definition | Storage
--------+---------+------------+---------
 id     | integer | id         | plain
unique, btree, for table "public.test", predicate (is_number_encrypt = false AND code = 2)
```

不是所有的条件比较多的 query 都适合建 Partial Indexes，主要合适场景为，where 中条件固定（有固有业务逻辑），且选择比较好的情况下。

9、Decompose DDL【分解 DDL】

```
例如 Add a column with not null and default value

alter table student add column test_col char(8) not null DEFAULT 'hello';
ALTER TABLE
Time: 3007667.459 ms

此种操作的运行时间,在生产上是无法接受的!

分解执行
1.  add column
alter table student add column test_col char(8);
ALTER TABLE
Time: 0.680 ms

2.  set default
alter table student alter COLUMN test_col SET DEFAULT 'hello';
ALTER TABLE
Time: 11.862 ms

3. update
update table student set test_col=DEFAULT;
change to
update table student set test_col=DEFAULT where id in (select id from student where test_col is null limit 5000); \watch 3

此步会多次执行,但是每次update 仅对5000条rows 加 FOR UPDATE这种Row-level 的lock,
对DBMS正常的并发产生的冲击很小, 且总时间未必多于原query,是一个在生产环境中非常实用的运维技巧

4. set not null
alter table student alter COLUMN test_col SET not null ;
Time: 20.662 ms
```

很多 DDL 操作，需要对 table 加 access exclusive 这种高粒度的 table-level locks，权衡利弊，可以将其分解执行，以底粒度的 lock 及稍长的执行时间替换高粒度 lock

10、Comprehensive optimization【综合优化】

```
UPDATE test_team SET status = 0 WHERE status = 1 AND (takeoff_date =  current_date  or available_count = 0);
Total runtime: 702526.769 ms

select count(1),
       sum(case when status=1 and takeoff_date = date'today' then 1 else 0 end),
       sum(case when status=1 and available_count = 0        then 1 else 0 end)
from test_team;
  count   |   sum   | sum
----------+---------+-----
 78567275 | 117311| 125

create index CONCURRENTLY ON test_team  (status, available_count) where status=1 and available_count = 0;
UPDATE test_team SET status = 0 WHERE (status = 1 AND takeoff_date =  current_date) or (status = 1 and available_count = 0);
                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------
 Update on test_team  (cost=2877.81..346942.03 rows=113046 width=260) (actual time=331.577..331.577 rows=0 loops=1)
   ->  Bitmap Heap Scan on test_team  (cost=2877.81..346942.03 rows=113046 width=260) (actual time=331.575..331.575 rows=0 loops=1)
         Recheck Cond: (((status = 1) AND (takeoff_date = ('now'::cstring)::date)) OR ((status = 1) AND (available_count = 0)))
         Filter: ((takeoff_date = ('now'::cstring)::date) OR (available_count = 0))
         ->  BitmapOr  (cost=2877.81..2877.81 rows=113051 width=0) (actual time=47.052..47.052 rows=0 loops=1)
               ->  Bitmap Index Scan on test_team_status_takeoff_date_idx1  (cost=0.00..2817.15 rows=112456 width=0)
                                                                          (actual time=47.040..47.040 rows=247037 loops=1)
                     Index Cond: ((status = 1) AND (takeoff_date = ('now'::cstring)::date))
               ->  Bitmap Index Scan on test_team_status_available_count_idx  (cost=0.00..4.13 rows=594 width=0)
                                                                            (actual time=0.011..0.011 rows=136 loops=1)
                     Index Cond: ((status = 1) AND (available_count = 0))
 Total runtime: 1331.644 ms

\di+ test_team*
 Schema |                    Name                      | Type  |  Owner   |   Table    |  Size   | Description
--------+----------------------------------------------+-------+----------+------------+---------+-------------
 public | test_team_status_takeoff_date_idx1           | index | postgres | test_team  | 2345 MB |
 public | test_team_status_available_count_idx         | index | postgres | test_team  | 64 kB   |
进一步优化的方法
update test_team set status = 0 where status=1 and available_count = 0 ;
update test_team set status = 0 where id in ( select id from test_team where status=1 and takeoff_date =date'today' limit 1000 ); \watch 3
```

一个既是 Multicolumn 又是 Partial Index，且分解执行的例子

11、索引的创建

索引并不是越多越好，存储索引本身也有空间开销，扫描索引本身也有时间开销，索引固然可以提高相应的query(不限于select)的执行效率，但同时也可能降低写入的效率，所以是否需要建index，如何建index需视具体情况而定，而且index也需要定期维护。

在生产实例上建 index，一定要使用 concurrently 参数，这种方式会以增量的方式建立 index，lock 粒度很低，不会阻塞写入数据。

各自适用范围简要说明:

◆B-tree: 最常用的index, 适合处理等值及范围queries.

◆Hash: 只能处理简单等值queries, 但由于**Hash index**的更改无法写入**WAL**, 所以一旦实例崩溃重启, 可能需要reindex或重建, 特别是有Primary/Standby 结构的集群中, 禁止使用Hash index.

◆GiST: 不是一种的简单index类型, 而是一种架构,可以在这种架构上实现很多不同的index策略. PostgreSQL 中的几何数据类型有很多GiST操作符类.

◆SP-GiST: GiST的增强, 引入新的index算法提高GiST在某些情况下的性能.

◆GIN: 反转index, 又称广义倒排index, 它可以处理包括多个键的值, 如数组等.

◆BRIN: 索引适用于数据值分布和物理值分布相关性很好的情况.

12、查找需要删除的索引

```
select
        pi.schemaname, pi.relname, pi.indexrelname, pg_size_pretty(pg_table_size(pi.indexrelid)),idx_scan,idx_tup_read,idx_tup_fetch
from
        pg_indexes pis
join
        pg_stat_user_indexes pi
  on
        pis.schemaname = pi.schemaname
        and pis.tablename = pi.relname
        and pis.indexname = pi.indexrelname
left join
        pg_constraint pco
  on
        pco.conname = pi.indexrelname
        and pco.conrelid = pi.relid
where
            pi.schemaname='public'
        and pco.contype is distinct from  'p' and pco.contype is distinct from  'u'
        and (idx_scan,idx_tup_read,idx_tup_fetch) = (0,0,0)
        and pis.indexdef !~ ' UNIQUE INDEX '
order by
        pg_table_size(indexrelid) desc limit 1
;
-[ RECORD 1 ]-+----------------------------------------
schemaname     | public
relname        | student
indexrelname   | student_product_id_idx
pg_size_pretty | 2543 MB
idx_scan       | 0
idx_tup_read   | 0
idx_tup_fetch  | 0
```

13、查找重复的索引

```
Indexes:
    "student_begin_time_idx" btree (begin_time)
    "student_begin_time_idx1" btree (begin_time)

select
        *
from
        (
        select
                tablespace,
                schemaname,
                tablename,
                indexname,
                pg_size_pretty(pg_table_size(schemaname||'."'||indexname||'"')) as index_size,
                indexdef,
                count(1) over
                ( partition by schemaname, tablename, regexp_replace(indexdef, E'(INDEX )(.+)( ON )(.+)',E'\\1\\3\\4') )
        from
                pg_indexes
        ) as foo
where
        count > 1
;
-[ RECORD 1 ]-----------------------------------------------------------------
tablespace  |
schemaname  | public
tablename   | student
indexname   | student_begin_time_idx1
index_size  | 115 MB
indexdef    | CREATE INDEX student_begin_time_idx1 ON student USING btree (begin_time)
count       | 2
-[ RECORD 2 ]-----------------------------------------------------------------
tablespace  |
schemaname  | public
tablename   | student
indexname   | student_begin_time_idx
index_size  | 428 MB
indexdef    | CREATE INDEX student_begin_time_idx ON student USING btree (begin_time)
count       | 2
```

14、查找需要维护的索引，并自定创建索引维护 SQL

Reference information 1:

```
select * from pgstattuple('student_name_idx');
-[ RECORD 1 ]------+----------
table_len          | 392134656
tuple_count        | 6242360
tuple_len          | 227570264
tuple_percent      | 58.03
dead_tuple_count   | 0
dead_tuple_len     | 0
dead_tuple_percent | 0
free_space         | 137661492
free_percent       | 55.11
```

Reference information 2:

```
仅适合Btree Index https://github.com/pgexperts/pgx_scripts/blob/master/bloat/index_bloat_check.sql

 db_name | schema_name | table_name | index_name | bloat_pct | bloat_mb | index_mb | table_mb | index_scans
---------+-------------+------------+------------+-----------+----------+----------+----------+-------------
 mydb    | public      | table_a    | index_a    |        89 |      303 |  340.688 |  373.844 | 364957
 mydb    | public      | table_b    | index_b    |        88 |      291 |  328.906 |  373.844 | 0
 mydb    | public      | table_c    | index_c    |       100 |      271 |  271.313 |    1.094 | 89489443
(3 rows)
```

```
select flag,
        CASE
                WHEN flag = 1 THEN
                        CASE
                        WHEN indexdef !~ ' WHERE ' THEN
                                regexp_replace(indexdef, E'(INDEX )(.+)( ON )(.+\\)$)'  ,E' \\1 CONCURRENTLY \\3 \\4 TABLESPACE  pg_default ','g') ||'; '
                        ELSE
                                regexp_replace(indexdef, E'(INDEX )(.+)( ON )(.+)( WHERE )'  ,E' \\1 CONCURRENTLY \\3 \\4 TABLESPACE  pg_default \\5 ','g') ||'; '
                        END
                WHEN flag = 2 THEN
                                'ANALYZE VERBOSE '||tablename||E' ; \nselect pg_sleep(500);\nDROP INDEX CONCURRENTLY IF EXISTS '||indexname||E';\n'
        END as SQL
    from
        (
        select
                generate_series(1,2) as flag,
                indexdef,
                indexname,
                tablename
        from
                pg_indexes pi
        join
                pg_namespace n
          on
                pi.schemaname = n.nspname
        join
                pg_class pcl
          on
                pcl.relnamespace = n.oid
                and pcl.relname = pi.tablename
        left join
                pg_constraint pco
          on
                pco.conname = pi.indexname
                and pco.conrelid = pcl.oid
        where
                (pi.schemaname, pi.tablename, pi.indexname) in (select * from reindex_tmp) --reindex_tmp is a temporary table based on the above reference informations.
                and pco.contype is distinct from  'p' and pco.contype is distinct from  'u'
        order by
                        tablename, indexname, pg_table_size(indexname::text) desc, flag asc
        ) as foo
    order by
                tablename, indexname, pg_table_size(indexname::text) desc, flag asc
limit 2 ;
 flag |                                             sql
------+------------------------------------------------------------------------------------------------
    1 | CREATE UNIQUE  INDEX  CONCURRENTLY  ON  student  USING btree (a, b) TABLESPACE  pg_default ;
    2 | ANALYZE VERBOSE student ;
      | select pg_sleep(500);
      | DROP INDEX CONCURRENTLY IF EXISTS student_a_b_idx;
      |
(2 rows)
```

15、一个 index size 影响 query plan 的例子

```
优化前,发现有Primary key，却使用其他index

  Index "public.test_id_settle_price_idx"
    Column     |  Type    |  Definition   | Storage
---------------+----------+---------------+---------
 id            | integer  | id            | plain
 settle_price  | bigint   | settle_price  | plain
unique, btree, for table "public.test"

      Index "public.test_pkey"
 Column |  Type    | Definition | Storage
--------+----------+------------+---------
 id     | integer  | id         | plain
primary key, btree, for table "public.test"

explain analyze select id, display_id, product_id from test where id = 211477920;
                                              QUERY PLAN
----------------------------------------------------------------------------------------------
 Index Scan using test_id_settle_price_idx on test  (cost=0.43..8.45 rows=1 width=23)
                                                     (actual time=0.014..0.014 rows=1 loops=1)
   Index Cond: (id = 211477920)
 Planning time: 0.158 ms
 Execution time: 0.029 ms
(4 rows)
```

分析原因：首先，看index size

```
                              List of relations
 Schema |           Name            | Type  |  Owner   | Table  |  Size  | Description
--------+---------------------------+-------+----------+--------+--------+-------------
 public | test_id_settle_price_idx  | index | postgres | test   | 395 MB |
 public | test_pkey                 | index | postgres | test   | 741 MB |
```

再看share_buffer 中使用量

```
create extension pg_buffercache ;

SELECT
c.relname,
pg_size_pretty(count(*) * (select setting from pg_settings where name='block_size')::integer ) as buffered,
round(100.0 * count(*) /
(SELECT setting FROM pg_settings
WHERE name='shared_buffers')::integer,1)
AS buffers_percent,
round(100.0 * count(*) * (select setting from pg_settings where name='block_size')::integer /
pg_relation_size(c.oid),1)
AS percent_of_relation
FROM pg_class c
INNER JOIN pg_buffercache b
ON b.relfilenode = c.relfilenode
INNER JOIN pg_database d
ON (b.reldatabase = d.oid AND d.datname = current_database())
WHERE c.relname in ('test_id_settle_price_idx','test_pkey')
GROUP BY c.oid,c.relname
;
          relname          | buffered | buffers_percent | percent_of_relation
---------------------------+----------+-----------------+---------------------
 test_id_settle_price_idx1 | 132 MB   |             0.4 |                33.4
 test_pkey                 | 2400 kB  |             0.0 |                 0.3
```

可见share buffer 中 test_id_settle_price_idx 的比例多么大，
原因是由于test_pkey膨胀，查询规化器计算cost后放弃使用pkey上的index，
而使用test_id_settle_price_idx，经过一些Query运行后，test_id_settle_price_idx缓存在内存中的size逐渐升高

解决方案： 维护primary key

```
create unique index CONCURRENTLY on test (id);
alter table test drop constraint test_pkey;
alter table test add primary key using index test_id_idx;
alter table test rename CONSTRAINT test_id_idx to test_pkey;

 Schema |           Name            | Type  |  Owner   | Table  |  Size
--------+---------------------------+-------+----------+--------+--------
 public | test_id_settle_price_idx  | index | postgres | test   | 395 MB
 public | test_pkey                 | index | postgres | test   | 282 MB
```

再看share_buffer

```
          relname          | buffered | buffers_percent | percent_of_relation
---------------------------+----------+-----------------+---------------------
 test_id_settle_price_idx  | 1504 kB  |             0.0 |                 0.4
 test_pkey                 | 107 MB   |             0.3 |                37.8
(2 rows)
```

可以看到test_pkey维护后，其在share buffer的占比立即上升，且test_id_settle_price_idx 随之下降

```
explain analyze select id, display_id, product_id from test where id = 211477920;
                                   QUERY PLAN
-----------------------------------------------------------------------------------------
 Index Scan using test_pkey on test  (cost=0.43..8.45 rows=1 width=23)
                                     (actual time=0.013..0.014 rows=1 loops=1)
   Index Cond: (id = 211477920)
 Planning time: 0.155 ms
 Execution time: 0.030 ms
(4 rows)
```

可见，已使用了 test_pkey，由于Query本身很快，所以Query运行时间并无太大差别，
但是却节省了share buffer 26MB的内存
每个DB 节点(不仅仅Master，还有Slave) 节省了 741 - 282 = 459 MB 的磁盘空间
由此可见，对index的维护是很有意义的.

16、一些影响 query plan 的参数（parameters）

```
enable_xxx

select name, setting from pg_settings where name ~ 'enable';
          name          | setting
------------------------+---------
 enable_bitmapscan      | on
 enable_hashagg         | on
 enable_hashjoin        | on
 enable_indexonlyscan   | on
 enable_indexscan       | on
 enable_material        | on
 enable_mergejoin       | on
 enable_nestloop        | on
 enable_seqscan         | on
 enable_sort            | on
 enable_tidscan         | on
(11 rows)
```

开/关这些参数，可以人工干预查询规化器生成的Query Plan

`default_statistics_target`

此参数控制查询规划器所需的统计信息表中采样（数据分布直方图）行数.
其默认是100，重载的实例，可以将其调大，比如1000.
对于经常参与查询且再where 自己中频繁使用的列，可以考虑提升采样的行数.

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name table_name ALTER [ COLUMN ] column_name SET STATISTICS integer
```

`random_page_cost`

随机页访问成本比，简称RPC，它表示在磁盘上顺序读取和随机读取同一条记录的性能之比.
此参数默认值为4.0，一般在SSD上可以将其设置为2.0至2.5之间.
可以在DB Instance，Tablespace，单个DB 3个级别设置RPC.

```
更改postgresql。conf 中 random_page_cost 后 reload
ALTER DATABASE mydb set random_page_cost TO 2.0;
ALTER TABLESPACE pg_tbl SET (random_page_cost = 2.0);
```