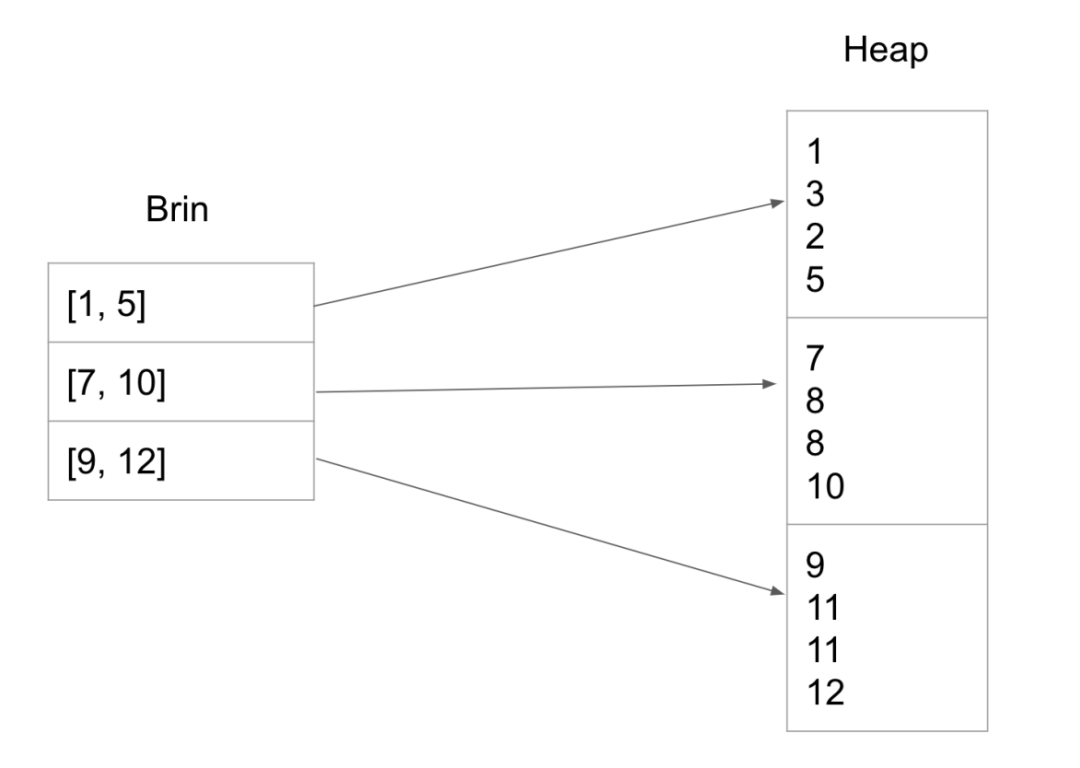
**BRIN索引介绍**

BRIN(Block Range Index)是块范围索引，索引记录的是存储数据块中的元组字段的最大值和最小值，用于过滤不符合条件的数据块。因此BRIN索引空间占用特别的小，同时建立起来也更快。BRIN是一个不准确的索引，因为它不包含表行的TID。

以下图为例，右边的堆表包含三个block，第一个block中有4个tuple，字段上的值分别是1、3、2、5。因此，与这个block相对应的Brin的元组就记录了 block的最小值1，最大值5。同理，7、8、8、10的最小值是7，最大值是10；9、11、11、12的最小值是9，最大值是12。这样就获得了一个 Brin。



在执行带有条件的查询时，就可以利用Brin记录的这些信息，找到相应的block，同时滤掉大部分block，从而起到Index的作用。

如果将每个范围视为“虚拟”分区，则可以将BRIN视为分区的替代方案。

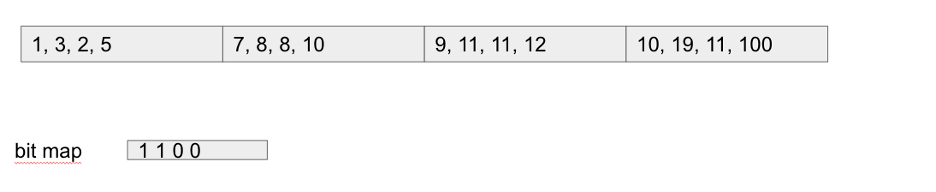
**索引扫描**

PostgreSQL提供了Bitmap Index Scan。Bitmap Index相对于Index Scan的劣势在于，它必须建立一个 bitmap, 而 bitmap会产生一些代价。但Bitmap Index的好处在于它把对于数据存储的随机读取变成了顺序读取，在数据量比较大的时候具有一定优势。

如果数据量大时，bitmap可能非常大。但bitmap是可以压缩的，压缩的方式也十分简单。当一个block被命中的Tuple比较多时，就可以直接用一个bit来代表整个block，从而把 bitmap压缩到很小。而由于Brin Index索引的单位是Block或一组Block，输出的Bitmap天然是压缩到最小的情况。

例如下面的例子中，得到的bitmap 就是1、1、0、0。该bitmap起到了索引应有的作用，4个block中过滤了2个block，因此只需要扫描2个block的数据。Brin Scan生成bitmap以后，就可以完全复用bitmap index scan的代码了。确实定位到了它们，则必须仔细检查所有块中的所有行，以在其中选择匹配的行。

select \* from t where a > 1 and a < 8;



**索引更新**

Brin在维护更新的逻辑较为简单。由于每个元组代表一组Block索引字段的最大、最小值，如果Insert或者Update新的数据超出了最大、最小值的范围，则更新元组；如果落在Brin的范围区间内，则不需要进行任何操作；

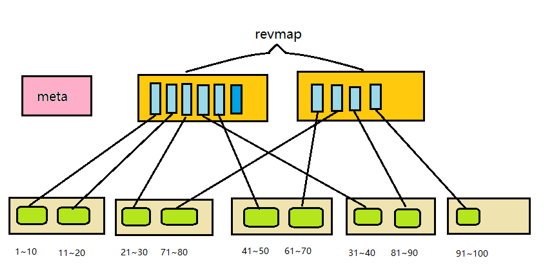
而Delete预期范围是缩小的，但很难有方式来知晓是否真的缩小，例如范围是1~5，删除4，范围没有缩小；但如果删除值为5的Tuple，由于可能存在很多个Tuple值均为5，很难知晓是否范围缩小了。因此在Delete的时候，不需要做任何操作。

随着Insert、Update的操作的执行，Brin的范围会慢慢变大，因此在整个更新的过程中可能会损失一些性能。但是由于brin index本身就是一个不追求精确的Index，在索引和Bitmap Index Scan之后，还要用查询条件进行过滤，因此不会产生错误结果。

BRIN对数据写入、更新、删除的影响很小。

**索引结构**

BRIN索引由三部分结构组成：meta page，revmap page和regular page。



meta page：包含元数据

revmap page(反向范围映射页面)：它是指向相应索引行的指针（TID）的数组

regular page：每个索引行都包含一个范围的最大值和最小值

**BRIN的优缺点**

1.优点

对于B-tree Index这样的密集索引，对应数据表的每一条数据，索引里也会存在一条数据，也就是说B-tree Index里的条目数目，大体上和数据表里的条目数是相当的。它的大小和数据表是一个量级的。如果表非常大，在这种表上创建B-树索引耗费的空间也非常大大，以至于无法全部容纳在内存中，这会导致内存和磁盘间的索引数据块换入换出，从而严重影响查询速度。但是Brin就完全不同了，数据表中的一个block或者一组block只会在Brin里生成一个条目，因此占用空间会非常小。

Brin的另一个优势是创建快。和B-tree Index一样，在创建brin的过程中，需要对表做一次完整的扫描。但是由于写需求很小，创建速度会快很多。

2.缺点

Brin Index只有在数据具有一定的分布特点时才有用。如果数据在索引字段上没有任何的分布特性，即它的空间分布和在索引字段上的值是没有任何关系，数据在空间上是完全随机分布的，这种情况下，Brin Index的效率会很低，因为能过滤掉的block将很少。

如果数据具有一定的分布特性，索引字段上的值和物理分布具有一定的关联，这种情况下，Brin的效率将很高。例如时序数据，在时间或序列字段创建BRIN索引，进行等值、范围查询时效果很好。

**BRIN适用场景**

表非常大

数据有一定的分布特性

用户不想在Index上付出太多存储空间

表不更新或很少更新，对于新插入的行会插入到表的末尾。

适用于数据仓库和分析报告。

**BRIN应用案例1-时间类型**

创建测试数据

create table t1(id uuid,name text,age int,join\_time timestamp);

insert into t1 select gen\_random\_uuid(),md5(random()::text),random()\*30,n from generate\_series('2022-01-01','2022-02-01',interval '1 second') n;

select \* from t1 where id='xxx';--200多MB数据，顺序扫描花了200多毫秒。

**思考：在哪些列上创建索引？**

数据必须在一定程度上与其物理位置相关，PostgreSQL收集表列统计信息，其中包括相关性的统计信息。优化器使用此值在常规索引扫描和位图扫描之间进行选择，可以使用它来估计BRIN索引的适用性。

示例中，数据是按时间排序的。因为将行添加到表中（没有删除和更新）时，它们在文件中的布局是一个接一个的。

analyze t1;

select attname, correlation from pg\_stats where tablename='t1' order by correlation desc;

该值接近正负1，创建BRIN索引最合适。

create index on t1 using brin(join\_time);

执行计划

explain analyze select \* from t1 where join\_time >= '2022-01-01 12:00:00' and join\_time <= '2022-01-01 13:00:00';

与B树比较

create index on t1(join\_time);

select pg\_size\_pretty(pg\_relation\_size(‘indexname'));

比BRIN大，但查询速度快

explain analyze select \* from t1 where join\_time >= '2022-01-01' and join\_time < '2022-01-02';

BRIN的优点:牺牲效率，但获得了很大的空间。

**参数调优**

BRIN索引有一个参数pages\_per\_range可以用来近一步提升BRIN索引的性能。

块范围的尺寸在索引创建时由 pages\_per\_range 存储参数决定，默认为128，表示每128个数据块统计一次边界，它影响BRIN索引的精确度和 BRIN索引的大小。 索引项的数量等于该关系的页面数 / pages\_per\_range。 因此， 该值越小， 索引会变得越大， 但是与此同时，存储的范围数据可以更加精确并且在索引扫描期间可以跳过更多数据块。

select max(join\_time)-min(join\_time) from t1;--得到天数

select relpages from pg\_class where relname ='t1';--得到页数

select relpages/days;--得到每条生成多少页

精度为1：

create index on t1 using brin(join\_time) with (pages\_per\_range=1);

explain analyze select \* from tab\_brin1 where c\_time between '2022-01-01 12:00:00' and '2022-01-01 13:00:00';

精度为50：

create index on t1 using brin(join\_time) with (pages\_per\_range=50);

explain analyze select \* from tab\_brin1 where c\_time between '2022-01-01 12:00:00' and '2022-01-01 13:00:00';

pages\_per\_range定义数据块的数量，为BRIN索引的每条记录统计的数据块范围。默认值为128。

如果这个值很大，则索引就会很小，索引扫描就会很迅速，但是后续内存中的Recheck就会很多，因为把大量的不相关数据拉到内存中了。

如果这个值很小，索引的过滤性越好，但索引也会越大。由于每筛选一次字段PostgreSQL 都要扫描全部的BRIN索引，所花费的时间也会变长，因此需要根据表的大小与应用场景去调整其值的大小。

**内部结构**

查看页面类型，总共3个页面

select \* from brin\_page\_type(get\_raw\_page('t1\_join\_time\_idx',0));

查看元页信息：range的大小以及为 revmap 分配了多少页：

select \* from brin\_metapage\_info(get\_raw\_page('t1\_join\_time\_idx',0));

查看revmap page信息。

select count(\*) from brin\_revmap\_data(get\_raw\_page('t1\_join\_time\_idx',1));

select \* from brin\_revmap\_data(get\_raw\_page('t1\_join\_time\_idx',1)) limit 5;

select \* from brin\_revmap\_data(get\_raw\_page('t1\_join\_time\_idx',1)) offset 236 limit 5;

select \* from brin\_revmap\_data(get\_raw\_page('t1\_join\_time\_idx',1)) offset 1354;

1. 从pg\_class中知道，t1表有30437个heap page。t1\_join\_time\_idx这个BRIN索引是缺省按照pages\_per\_range=128来创建的。从这些信息可以计算出，需要ceiling(30437/128::decimal)=238个索引条目。这和我们从revmap从获得的数据是匹配的；

2. 因为一个regular page可以存放n个索引条目，所以238个条目需要1个 regular  page来存放。也和revmap page中获得的数据一致；

3. 由于一个revmap page可以存放1360个索引条目的位置信息，因此对于一个revmap page来讲，可以管理1360\*128(pages\_per\_range)\*8KB(page size)/1024=1360MB的表。当前t1表的大小是238MB，只要表小于1360MB，一个revmap page就足够了；

查看regular page内容：

select count(\*) from brin\_page\_items(get\_raw\_page('t1\_join\_time\_idx',2),'t1\_join\_time\_idx');

select \* from brin\_page\_items(get\_raw\_page('t1\_join\_time\_idx',2),'t1\_join\_time\_idx') limit 5;

select \* from brin\_page\_items(get\_raw\_page('t1\_join\_time\_idx',2),'t1\_join\_time\_idx') offset 236 limit 5;

1. blknum是一个索引条目的起始heap page编号。例如第一个条目涵盖从0到127的heap page，所以blknum等于0；

2. attnum是指一个条目中存放的是第几个字段的信息。本例中，由于只索引了一个字 段join\_time，所以attnum全部为1。如果索引两个字段，会有attnum为2的情况；

3. allnulls是告诉我们在一个条目涵盖的128个heap page对应的索引字段上是否全部为NULL值。本例因为全部有值，所以allnulls都是false；

4. hasnulls顾名思义，告诉我们在一个条目涵盖的128个heap page对应的索引字段上是否含有NULL值。本例因为全部有值，所以allnulls都是false；

5. value这个字段可以说是BRIN中最重要、最精华的部分，它存储了一个条目涵盖的128个heap page对应字段的最小值和最大值。PG就是根据这个value值来判断SQL的执行器（executor）是否需要扫描这128个heap page。拿第一个条目来说明，，当执行例如joid\_time='2022-01-01 01:00:00' 的查询过滤时，PG只需要扫描第1索引条目涵盖的128个heap page就可以，从而提升查询效率。

**BRIN应用案例2-数值类型**

创建测试数据

create table t2(id serial,name text,age int);

insert into t2(name,age) select md5(n::text),random()\*30 from generate\_series(1,1000000) n;

select \* from t2 where id='10000';--70多MB数据，顺序扫描花了90多毫秒。

select attname, correlation from pg\_stats where tablename='t1' order by correlation desc;

create index on t2 using brin(id);

select \* from t2 where id='10000';--查找2毫秒左右

create index on t2(id);

select pg\_size\_pretty(pg\_relation\_size(‘indexname'));

查看页面类型，总共3个页面

select \* from brin\_page\_type(get\_raw\_page('t2\_id\_idx',0));

查看元页信息：range的大小以及为 revmap 分配了多少页：

select \* from brin\_metapage\_info(get\_raw\_page('t2\_id\_idx',0));

查看revmap page信息。

select count(\*) from brin\_revmap\_data(get\_raw\_page('t2\_id\_idx',1));

select \* from brin\_revmap\_data(get\_raw\_page('t2\_id\_idx',1)) limit 5;

查看regular page信息：

select \* from brin\_page\_items(get\_raw\_page('t2\_id\_idx',1),'t2\_id\_idx') limit 1;