**HASH索引介绍**

HASH索引是使用hash函数对索引的关键字进行hash运算后，将hashcode作为键存放，它只能处理简单的等值查询。其主要目的就是对于某些数据类型（索引键），快速找到匹配的行的ctid。

hash索引特别适用于字段值非常长（不适合b-tree索引，因为b-tree一个PAGE至少要存储3个ENTRY，所以不支持特别长的VALUE）的场景，例如很长的字符串，并且用户只需要等值搜索，建议使用hash index。

在pg10之前是不提倡使用hash索引的，因为hash索引不会写wal日志，无法保证持久性，在高可用场景下可能给出错误的查询结果。不过从pg10开始解决了这一问题，并且对hash索引进行了一些加强。

通常，数据类型的允许值范围非常大：在 text 类型的列中我们可以任意多个字符，在varchar类型中最大可以存放1GB。但通常情况下，不会存放那么多。

哈希的思想是将少量的数字（从0到N -1，总共N个值）与任何数据类型的值相关联。通过哈希函数所获得的数字可用作常规数组的索引，其中将存储对表行（TID）的引用。该数组的元素称为哈希表存储桶（bucket），如果相同的索引值出现在不同的行中，则一个存储桶可以存储多个TID。

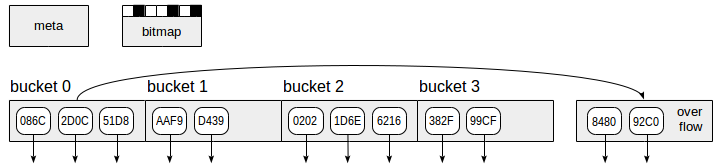
**索引结构**

当插入索引时，先计算键的哈希值。PostgreSQL中的哈希函数总是返回 integer 类型，其范围为2的32次方≈40亿个值。哈希桶的数量最初等于2，然后根据数据大小动态增加。哈希桶编号可以使用位算法从哈希码中计算出来。为了节省空间，哈希桶不是存储键，而是存储了键的哈希码和 TID。

当搜索索引时，我们用哈希函数计算键的哈希码并获取哈希桶编号。然后需要遍历存储桶的内容，并仅返回具有适当哈希码的匹配TID。由于存储的 hashcode - TID 对是有序的，因此可以高效地完成此操作。

但是，两个不同的键可能会进入一个哈希桶，而且可能会具有相同的哈希码，而且无法消除这种冲突。因此，访问方法要求通用索引引擎通过重新检查表行中的条件来验证每个TID（引擎可以在进行可见性检查的同时执行此操作）。

哈希索引包含4种页：meta page, primary bucket page,overflow page, bitmap page。



• 元页面(meta page 第0页)：包含有关索引的控制信息，指导如何找到其他页面（每个bucket的primary page）。

hash index将存储划分为多个bucket（逻辑概念），每个bucket中包含若干page（每个bucket的page数量不需要一致），当插入数据时，根据计算得到的哈希码，通过映射算法，映射到某个bucket，也就是说数据首先知道应该插入哪个bucket中，然后插入bucket中的primary page，如果primary page空间不足时，会扩展overflow page，数据写入overflow page。在page中，数据是有序存储（TREE），page内支持二分查找(binarysearch)，而page与page之间是不保证顺序的，所以hash index不支持order by。

• 哈希桶页面-索引的主页，将数据存储为“哈希码-TID”对。

• 溢出页面：是bucket里面的页，与哈希桶页面的结构相同，当primary page没有足够空间时，扩展的块称为overflow page。

• 位图页面：记录primary  page, overflow page是否为空可以被重用。

从索引页元素开始的向下箭头表示TID，即对表行的引用。

每次索引增加，PostgreSQL即时创建的哈希桶（也就是页面）数量是上次创建的存储桶数量的两倍。为了避免一次分配大量潜在的页面，版本10使大小增加更加平稳。对于溢出页面，它们会根据需要进行分配，并在位图页面中进行跟踪，位图页面也会根据需要进行分配。

注意：哈希索引不能减小大小。如果删除一些索引行，则已经分配的页面不会返回给操作系统，而只会在VACUUM之后重新用于新数据(通过bitmappage跟踪）。减小索引大小的唯一选项是使用REINDEX或VACUUM FULL命令从头开始重建索引。

创建哈希索引：

create index on t1 using hash(name);

explain  select \* from t1 where name = '1f0e3dad99908345f7439f8ffabdffc4';

哈希索引的通用性远低于“ B-tree”，其效率也值得怀疑。因此，它是适用场景很小。

哈希函数不保留顺序关系，生成的哈希码是离散的，所以无法对键本身排序。通常哈希索引可以支持唯一的操作“=”：因此，哈希索引不能返回有序数据（can\_order，orderable）。

出于同样的原因，哈希索引不能处理NULL：equals 操作对NULL（search\_nulls）没有意义。由于哈希索引不存储键（仅存储其哈希码），因此它不能用于仅索引访问（returnable）。此访问方法也不支持多列索引（can\_multi\_col）。

**内部结构**

查看元页面信息：

select hash\_page\_type(get\_raw\_page('idx\_t1\_name',0));--查看页面类型

select ntuples, maxbucket from hash\_metapage\_info(get\_raw\_page('idx\_t1\_name',0));--查看元页中索引行数和最大哈希桶数

查看哈希桶页面统计信息：

select hash\_page\_type(get\_raw\_page('idx\_t1\_name',1));--查看页面类型

select live\_items, dead\_items from hash\_page\_stats(get\_raw\_page('idx\_t1\_name',1));--查看哈希桶页面内活动元组和死元组的数量，即可以清除的元组的数量

查看HASH索引的桶页或者溢出页中存储的数据的信息：

select \* from hash\_page\_items(get\_raw\_page('idx\_t1\_name',2));

查看HASH索引的溢出页面的位状态：

SELECT \* FROM hash\_bitmap\_info(get\_raw\_page('idx\_t1\_name',n));