**GIN应用案例1-全文检索**

**gin索引介绍**

GIN(Generalized Inverted Index, 通用倒排索引) 是一个存储对(key, position list)集合的索引结构，如(‘hank’, ‘15:3 21:4’)中，表示hank在15:3和21:4这两个位置出现过。GIN的一个很好的类比是书末尾的索引，该索引为每个术语提供了该术语出现的页面列表。GIN索引通常用来处理如数组、JSONB、TSVECTOR等数据类型。其他一些扩展包（比如 pg\_trgm ）也会使用这种索引。  PostgreSQL的标准发布中包含了用于数组的GIN操作符类，比如， 它支持 @>、 <@、 =、 &&等操作符。

GIN索引的内部构造是提取出键的B-tree索引，叶子节点上每个元组是一个 postiog tree (同样是Btree结构)的指针或者一个 posting list (简单列表)。

键永远不会从GIN索引中删除。可以认为键可以消失、出现或变化，但是组成它们的键集合或多或少是稳定的。该解决方案极大地简化了多个进程使用索引并发工作的算法。

如果TID列表很小，则可以与该键放入同一页面。但是，如果列表很大，则需要更有效的数据结构，使用B树存储。这样的树位于单独的数据页上。

相同的键值只存储一次。

注意：与B树不同，GIN索引的页面是通过单向列表而不是双向列表连接的。

**示例：**

create table t2(info text, info\_tsv tsvector);

insert into t2(info) values

('Can a sheet slitter slit sheets?'),

('How many sheets could a sheet slitter slit?'),

('I slit a sheet, a sheet I slit.'),

('Upon a slitted sheet I sit.'),

('Whoever slit the sheets is a good sheet slitter.'),

('I am a sheet slitter.'),

('I slit sheets.'),

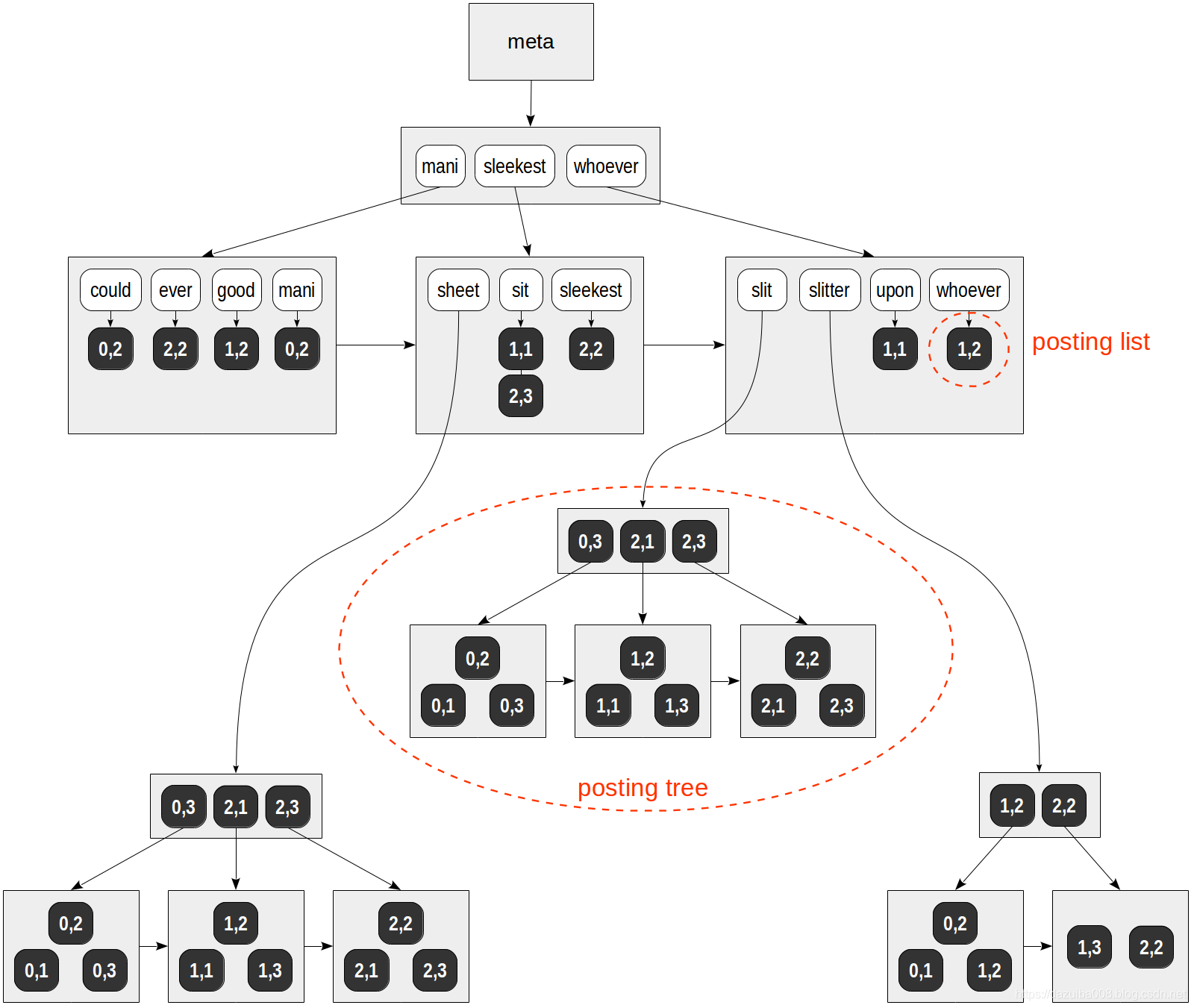
('I am the sleekest sheet slitter that ever slit sheets.'),

('She slits the sheet she sits on.');

update t2 set info\_tsv = to\_tsvector(info);

create index on t2 using gin(info\_tsv);

该GIN索引结构如下，黑色方块是TID编号，白色为单词,注意这里是单向链表，不同于B-tree的双向链表：



select ctid,\* from t2;

由上可见，sheet,slit,slitter出现在多行之中，所有会有多个TID，这样就会生成一个TID列表，并为之生成一棵单独的B-tree。

词素的TID列表已放入单独的B树中。

如何执行以下查询？

explain select info from t2 where info\_tsv @@ to\_tsquery('many & slitter');

首先从查询中提取单个词素：many 和 slitter。这是由to\_tsquery函数完成的，该函数考虑了运算符类决定的数据类型和策略：

select amop.amopopr::regoperator, amop.amopstrategy

from pg\_opclass opc, pg\_opfamily opf, pg\_am am, pg\_amop amop

where opc.opcname = 'tsvector\_ops'

and opf.oid = opc.opcfamily

and am.oid = opf.opfmethod

and amop.amopfamily = opc.opcfamily

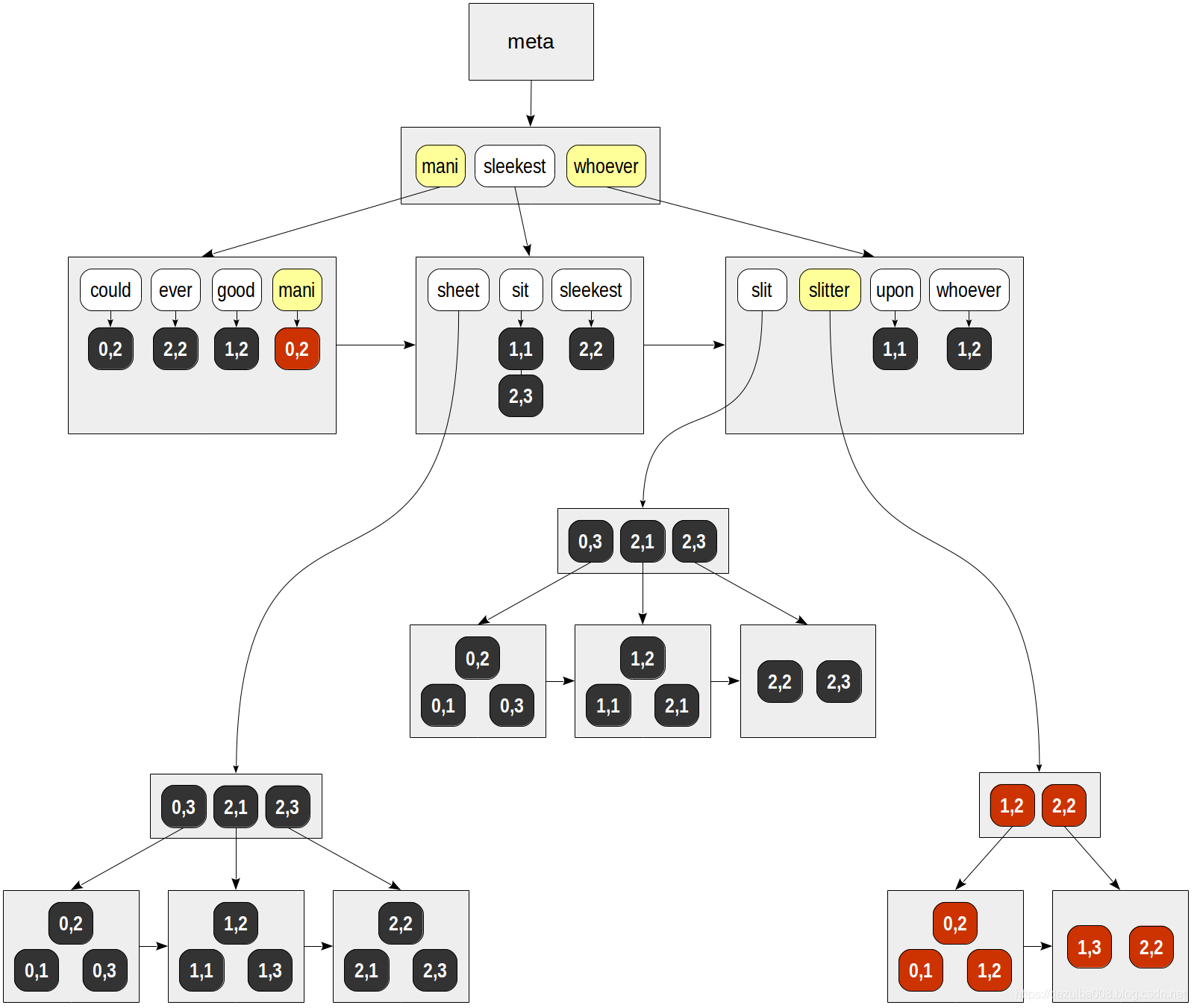
and am.amname = 'gin'

and amop.amoplefttype = opc.opcintype;

在词素的B树中，找到这两个键，并浏览TID列表。得到：

mani —（0,2）。

slitter —（0,1），（0,2），（1,2），（1,3），（2,2）。



最后，对于找到的每个TID，进行 and 连接，返回的唯一行是（0,2）：

结果是：

select \* from t2 where info\_tsv @@ to\_tsquery('many& slitter');

**部分匹配的搜索**

可以在全文搜索中使用部分匹配。例如：

select \* from t2 where info\_tsv @@to\_tsquery('slit:\*');

该查询将查找以 slit 开头的词素的文档。在此示例中，这样的词素是slit 和 slitter。

即使没有索引，查询也可以工作，但是GIN可以加快以下搜索的速度：

explain select info from ts where info\_tsv @@ to\_tsquery('slit:\*');

所有指定了前缀的词素都会在树中查找，并由 or 连接。

**GIN应用案例2-数组**

使用GIN的另一个示例是数组的索引。在这种情况下，数组元素进入索引，这可以加快对数组的许多操作：

select amop.amopopr::regoperator,amop.amopstrategy

from pg\_opclass opc, pg\_opfamily opf, pg\_am am, pg\_amop amop

where

opc.opcname = 'array\_ops' and

opf.oid = opc.opcfamily and

am.oid = opf.opfmethod and

amop.amopfamily = opc.opcfamily and

am.amname = 'gin' and

amop.amoplefttype = opc.opcintype;

创建测试数据：

create table t3(id int,name text,age int,score int[]);

insert into t3 select n,md5(n::text),random()\*10+20,array[random()\*100,random()\*100,random()\*100] from generate\_series(1,10000) n;

create index on t3 using gin(score);

explain  select \* from t3 where score = ARRAY[56,92,72];

该查询如何执行？

1. 从数组{56,92,72}中提取元素。得到56,92和72的值。

2. 在元素树中，找到提取的键，并为每个键选择TID列表。

3. 在找到的所有TID中，选择与运算符匹配的TID。对于运算符 =，只有那些TID与所有三个列表中出现的TID匹配（换句话说，数组必须包含所有元素）。但这还不够：数组还需要它不包含任何其他值，并且我们无法使用索引检查此条件。因此，在这种情况下，访问方法要求索引引擎重新检查与表一起返回的所有TID。

索引支持附加条件

create index on t3(age);

explain select \* from t3 where score = ARRAY[56] and age=22;

创建多列索引

create index on t3 using gin(score,age);  --ERROR

扩展 btree\_gin 添加了GIN运算符类来模拟常规B树的工作。

create extension btree\_gin;

create index on t3 using gin(score,age);

explain select \* from t3 where score = ARRAY[56] and age=22;

**GIN应用案例3-JSONB**

内置GIN支持的复合数据类型的另一个示例是JSONB。为了使用JSONB值，PG定义了许多运算符和函数，其中一些可以使用索引加快访问速度：

select opc.opcname, amop.amopopr::regoperator,amop.amopstrategy as str

from pg\_opclass opc, pg\_opfamily opf, pg\_am am, pg\_amop amop

where opc.opcname in ('jsonb\_ops','jsonb\_path\_ops')

and opf.oid = opc.opcfamily

and am.oid = opf.opfmethod

and amop.amopfamily = opc.opcfamily

and am.amname = 'gin'

and amop.amoplefttype = opc.opcintype;

默认情况下，使用第一个运算符类 jsonb\_ops。

示例：

create table t1(id int,name text,age int);

insert into t1 select n,md5(n::text),random()\*10+20 from generate\_series(1,10000) n;

create table t4(info jsonb);

insert into t4 select to\_jsonb(t1) from t1;

create index on t4 using gin(info);

查询执行计划

explain select \*  from t4 where info @> '{"name":"xxx"}';

**内部结构**

查询元页面信息：

select \* from gin\_metapage\_info(get\_raw\_page('t4\_info\_idx',0));

查看GIN索引数据：

select flags, count(\*) from generate\_series(0,2226) as g(id),gin\_page\_opaque\_info(get\_raw\_page('t4\_info\_idx',g.id)) group by flags;

select gin\_page\_opaque\_info(get\_raw\_page('t4\_info\_idx',n)) from generate\_series(0,2267) n;

{meta}：meta page

{}：internal page of element B-tree

{leaf}：leaf page of element B-tree

{data}：internal page of TID B-tree

{data,leaf,compressed}：leaf page of TID B-tree

查看页面（data,leaf,compressed）上TID的信息：

select \* from gin\_leafpage\_items(get\_raw\_page('t4\_info\_idx',10));

SELECT first\_tid, nbytes, tids[0:5] AS some\_tids FROM gin\_leafpage\_items(get\_raw\_page('t4\_info\_idx', 10));

注意：TID树的叶子节点实际上包含指向表行的小型压缩指针列表，而不是单个指针。

**GIN索引优化**

**1.更新优化**

GIN索引中的数据插入或更新非常慢。每个文档通常包含许多要索引的词素。因此，当添加或更新一个文档时，必须大量更新索引树。

如果同时更新多个文档，则它们的某些词素可能是相同的，总工作量将小于逐个更新文档时的工作量。

GIN索引提供了一个参数(fastupdate)，可以在创建索引时指定它，并在稍后更新：

create index on t2 using gin(info\_tsv) with (fastupdate = true);

启用此参数后，更新将累积在单独的无序列表（pending list）中。当此列表达到一定的大小或在清理期间，所有累积的更新将立即作用于索引。列表的大小由 gin\_pending\_list\_limit 参数决定。

但是这种方法有缺点：

首先，搜索速度变慢（因为除了树之外还需要浏览无序列表）。

其次，如果无序列表溢出，则下一次更新可能会意外地花费大量时间。

**2.限制查询结果**

GIN访问方法的一个特点是，结果总是以位图的形式返回，所有查询计划都使用位图扫描。

因此，使用LIMIT子句限制索引扫描结果的效率不是很高。：

explain select \* from t2 where info\_tsv @@ to\_tsquery('slit') limit 1;

估计成本为12.51，比构建整个位图的成本8.25稍大。

如果查询返回的结果集特别大，可以用参数 gin\_fuzzy\_search\_limit 来限制返回的行数，默认情况下等于零（没有限制）。一般建议设置为5000~20000比较合适。

设置阈值：

set gin\_fuzzy\_search\_limit = 1000;

select count(\*) from t2 where info\_tsv @@ to\_tsquery('slit');

set gin\_fuzzy\_search\_limit = 10000;

select count(\*) from t2 where info\_tsv @@ to\_tsquery('slit');

查询返回的行数是不同的。但限制并不严格：有时可能会返回多于指定的行数，参数名称中的 fuzzy 就是这个意思。

**3.频繁和不频繁的词素**

如何找出包含一个词素的文档数量？

select (unnest(info\_tsv)).lexeme, count(\*) from t2 group by 1 order by 2 desc;--小表，直接查询

思考：许多文档中出现相同的词素。使用 unnest 的查询将无法处理如此大的数据量，可以使用 ts\_stat 函数，该函数提供有关词素，出现它们的文档数以及出现的总数信息。

select word, ndoc from ts\_stat('select info\_tsv from t2') order by ndoc desc;

选择 sheet(经常出现) 和 mani(很少出现)：

select word, ndoc from ts\_stat('select info\_tsv from t2') where word = 'mani';

是否有文档同时出现这两个词汇？

select count(\*) from t2 where info\_tsv @@to\_tsquery('sheet & mani');  --t

如果获得两个词素的TID列表，然后进行 and 连接，这样的搜索显然效率低下，因为 sheet 必须遍历9个值，mani是1个值。

可以使用统计信息，统计信息显示 sheet 词素经常出现，而 mani 则很少出现。因此，先执行不经常使用的词素的搜索，然后从两个文档中是否存在sheep 词素。

\timing on

select count(\*) from t2 where info\_tsv @@ to\_tsquery('sheet & mani');

仅搜索 sheep 会花费更长的时间：

select count(\*) from t2 where info\_tsv @@to\_tsquery('sheet');

**GIST和GIN对比**

GIN索引紧凑性表现良好。

首先，如果在多个文档中出现相同的词素，通常它仅在索引中存储一次。

其次，TID以有序的方式存储在索引中，能够使用简单的方式来进行压缩。

创建B树索引，字段大小不能超过索引页的1/3：

create index on t2(info);

创建GiST索引：

create index on t2 using gist(info\_tsv);

查看索引大小：

select

pg\_size\_pretty(pg\_relation\_size('t2\_info\_tsv\_idx')) as gin,

pg\_size\_pretty(pg\_relation\_size('t2\_info\_tsv\_idx2')) as gist,

pg\_size\_pretty(pg\_relation\_size('t2\_info\_idx')) as btree;

由于表示的紧凑性，可以在从Oracle迁移过程中使用GIN索引来代替位图索引。通常，位图索引用于唯一值很少的字段，这对于GIN也非常有用。而且，PostgreSQL可以基于任何索引（包括GIN）动态构建位图。

**思考：选择GiST or GIN？**

对于许多数据类型(tsvector、pg\_trgm)，GiST和GIN都可以使用相同运算符类，那么选择哪种索引更合适？

1.GIN在准确性和搜索速度上均胜过GiST。如果数据更新不频繁并且需要快速搜索，则建议选择GIN。

2.如果对数据进行密集更新，则更新GIN的开销成本可能太大，此时建议选择GiST。

需要从这两个方面考虑，根据索引特征更好地平衡选择。

一般来说，GIN索引对静态数据是最好的，因为查找速度很快。对于动态数据， GiST索引更新比较快。

对于上面同时提供GIN和GIST索引的数据类型，建议优先使用GIN做索引。GiST索引往往对键做了有损压缩，查询性能不太好预测，某些场景下可能效果很差。

GIN索引对于插入更新操作效率比较低，如果要向一张表中插入大量数据， 最好先把GIN索引删除， 然后插入数据， 最后再把GIN索引重新建起来。把maintenance\_work\_mem参数调大， 可以更快地完成GIN索引的创建工作。

关于全文检索的GIN和GIST对比测试:

GIN索引查找比GiST快约三倍

GIN索引建立比GIST需要大约三倍的时间。

GIN索引更新比GiST索引速度慢，但如果快速更新支持无效，则慢了大约10倍

GIN索引比GiST索引大两到三倍