**RUM介绍**

尽管GIN具有强大的功能，但在某些方面RUM更胜一筹，RUM被称为下一代GIN，也叫高速全文检索。

大致来说，RUM可以视为GIN，但它另外存储位置信息，并可以按所需顺序返回结果（类似GiST的近邻搜索）。

RUM索引的缺点是构建和插入时间比GIN索引慢。 这是因为需要存储除键值之外的其他信息，并且RUM使用通用WAL记录。

**RUM常用操作符**

|  |  |  |
| --- | --- | --- |
| 操作符 | 返回值数据类型 | 描述 |
| tsvector <=> tsquery | float4 | 返回tsvector与tsquery之间的距离。 |
| timestamp <=> timestamp | float8 | 返回两个时间戳之间的距离。 |
| timestamp <=| timestamp | float8 | 只返回左侧时间戳的距离。 |
| timestamp |=> timestamp | float8 | 只返回右侧时间戳的距离。 |

**GIN的局限性**

1.排序慢：tsvector 数据类型包含词素，还包含它们在文档中位置的信息，有了有关词素的位置信息才能进行排序。但是GIN索引不存储词素的位置信息，因此在索引扫描之后，需要额外的扫描来检索词素位置。

2.短语查询慢：GIN索引需要位置信息来执行短语搜索，因此GIN索引无法有效支持短语搜索操作。

PostgreSQL中可以使用排名函数 ts\_rank 和 ts\_rank\_cd 为结果进行排名，但这会很慢。

示例：

create table t1(info text, info\_tsv tsvector);

insert into t1(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 t1 set info\_tsv = to\_tsvector(info);

create index on t1 using gin(info\_tsv);

SELECT ts\_rank\_cd (to\_tsvector('english','in the list of stop words'), to\_tsquery('list & stop'));--查看距离

select \* from t1 where info\_tsv @@ to\_tsquery('slit & slitter') order by ts\_rank\_cd(info\_tsv, to\_tsquery('slit & slitter') );--排序查询

select to\_tsvector('in the list of stop words') @@ to\_tsquery('list <2> stop');--查看词距

select \* from t1 where info\_tsv @@ to\_tsquery('slitter <-> slit');

**短语搜索**

全文搜索查询可以包含使用词素之间距离的特殊运算符：

select to\_tsvector('Clap your hands, slap your thigh') @@ to\_tsquery('hand <3> thigh');--true

可以指定单词必须连续：

select to\_tsvector('Clap your hands, slap your thigh') @@ to\_tsquery('hand <-> slap');--true

GIN索引可以返回包含这两个词素的文档，但只能通过查看tsvector来检查它们之间的距离：

select to\_tsvector('Clap your hands, slap your thigh');

**RUM应用案例**

在RUM索引中，每个词素不只是引用表中的行的TID，每个TID还提供了该词素在文档中出现的位置列表。

export PATH=/usr/local/pgsql/bin:$PATH

git clone https://gitee.com/mirrors\_postgrespro/rum.git

cd rum

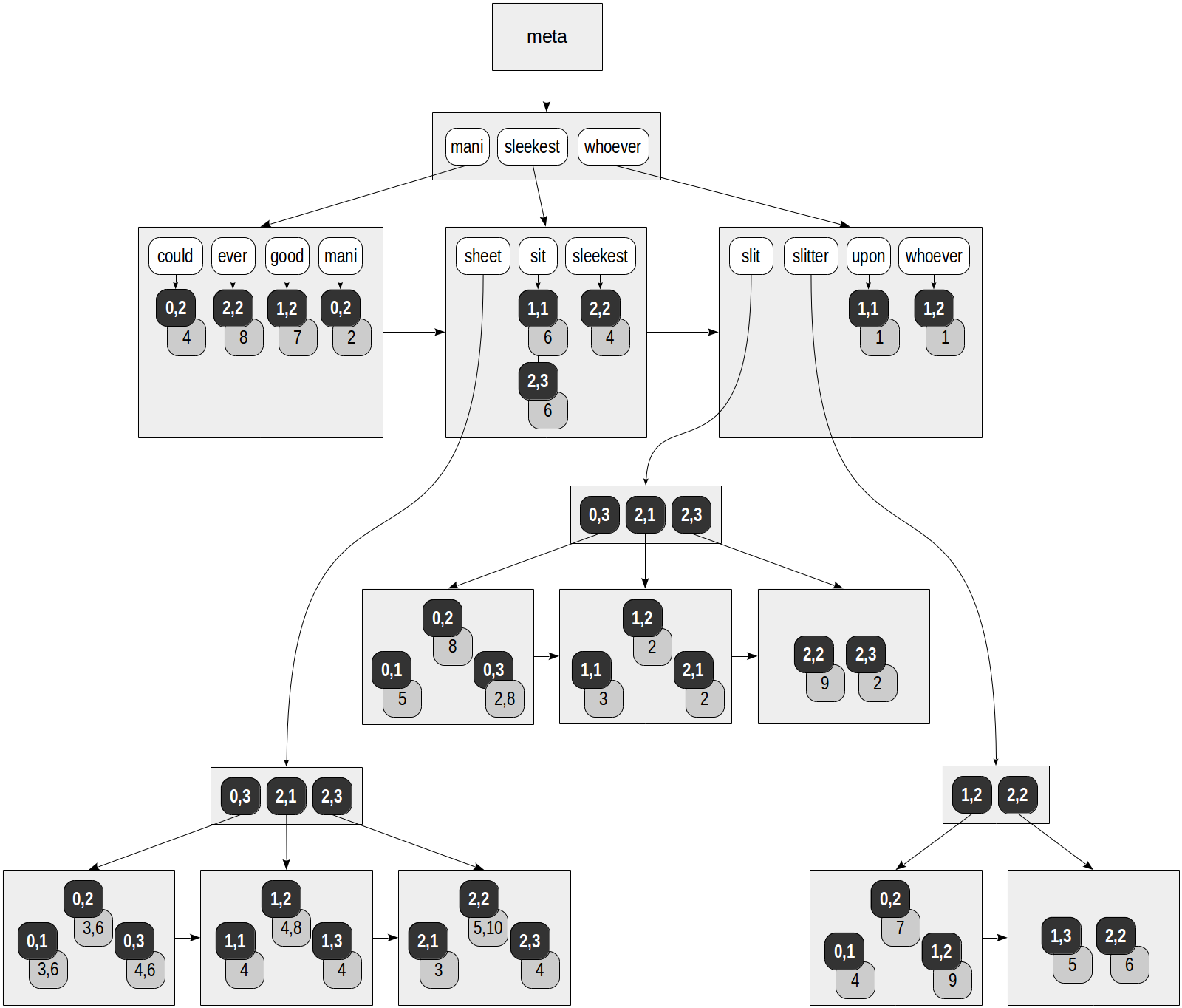
make USE\_PGXS=1

make USE\_PGXS=1 install

create extension rum;

create index on t1 using rum(info\_tsv);

图中的灰色方块包含添加的位置信息：



使用GIN索引执行使用短语搜索的查询：

create index on t1 using gin(info\_tsv);

explain analyze select \* from t1 where info\_tsv @@ to\_tsquery('slitter <-> slit');

从执行计划中可以看到，使用了GIN索引，但它 Bitmap Index Scan 返回4行，但最好返回2行，有2行在重新检查阶段被删除。

创建RUM

drop index t1\_info\_tsv\_idx;

create index on t1 using rum(info\_tsv);

索引现在包含了所有必要的信息，并且可以准确地执行搜索：

explain analyze select \* from t1 where info\_tsv @@ to\_tsquery('slitter <-> slit');

**按相关性排序**

RUM索引支持排序运算符，<=>，它返回文档（“tsvector”）和查询（“tsquery”）之间的距离。

select to\_tsvector('Can a sheet slitter slit sheets?') <=> to\_tsquery('slit');--16.44934

select to\_tsvector('Can a sheet slitter slit sheets?') <=> to\_tsquery('sheet');--13.159472

比较GIN和RUM：选择2个最相关的文档：

使用GIN索引：

explain analyze select \* from t1 where info\_tsv @@ to\_tsquery('slit & slitter') order by ts\_rank\_cd(info\_tsv,to\_tsquery('slit & slitter'))

limit 2;--有 sort 操作

使用RUM index，不需要查看额外的文档，也不需要单独排序：（因为是精确查询，所以不需要重新检查；因为rum索引会根据"附加信息"——距离来组织索引项，所以不需要排序）

explain analyze select \* from t1 where info\_tsv @@ to\_tsquery('slit & slitter') order by info\_tsv <=> to\_tsquery('slit & slitter') limit 2;

**附加信息**

RUM和GIN都可以创建多列索引，但是GIN存储每一列的词素独立于另一列的词素，而RUM能够将主字段与另一个字段“关联”（就是根据另一个可排序的字段来组织索引项，默认是使用距离排序）。为此，创建索引时需要指定操作符类 rum\_tsvector\_addon\_ops：

示例：

create table t1(info text,create\_time timestamp,info\_tsv tsvector);

insert into t1(info,create\_time) values

('Can a sheet slitter slit sheets?','2022-01-02'),

('How many sheets could a sheet slitter slit?','2022-03-10'),

('I slit a sheet, a sheet I slit.','2021-09-20'),

('Upon a slitted sheet I sit.','2020-01-01'),

('Whoever slit the sheets is a good sheet slitter.','2021-05-10'),

('I am a sheet slitter.','2018-01-31'),

('I slit sheets.','2022-10-15'),

('I am the sleekest sheet slitter that ever slit sheets.','2022-12-01'),

('She slits the sheet she sits on.','2020-09-15');

update t1 set info\_tsv = to\_tsvector(info);

create index on t1 using rum(info\_tsv RUM\_TSVECTOR\_ADDON\_OPS, create\_time) WITH (ATTACH='create\_time', TO='info\_tsv');

查询按附加字段排序：

select \*,create\_time <=> '2022-01-01 12:00:00' from t1 where info\_tsv @@ to\_tsquery('slit') order by create\_time <=> '2022-01-01 12:00:00';

搜索尽可能接近指定日期的匹配行，无论是更早还是更晚。要获得严格在指定日期之前（或之后）的结果，需要使用<=|（或|=>）运算符。

查询只需通过简单的索引扫描即可执行：

explain select \*,create\_time <=> '2022-01-01 12:00:00' from t1 where info\_tsv @@ to\_tsquery('slit') order by create\_time <=> '2022-01-01 12:00:00';

如果没有字段关联的附加信息，对于类似的查询，必须对结果进行排序。

除了日期，还可以将其他数据类型的字段添加到RUM索引中。几乎所有基本类型都受支持。

**GIN和RUM对比(wal大小)**

由于RUM比GIN存储更多的信息，因此它具有更大的大小，这就是快速搜索的成本。

注意：RUM是扩展，也就是说，可以在不对系统核心进行任何修改的情况下安装RUM。

因此，大量更改RUM索引会生成比GIN大得多的日志记录（GIN不是扩展，而是核心的一部分，它自己管理日志）。

GIN：

select pg\_current\_wal\_lsn();

delete from t1 where ctid='(x,x)';

select pg\_current\_wal\_lsn();

select pg\_wal\_lsn\_diff(pg\_lsn,pg\_lsn)

RUM：

select pg\_current\_wal\_lsn();

delete from t1 where ctid='(x,x)';

select pg\_current\_wal\_lsn();

select pg\_wal\_lsn\_diff(pg\_lsn,pg\_lsn)