**索引介绍**

索引是数据库中的一种快速查询数据的方法，就好比是一本书前面的目录，通过目录中页码就能快速定位到我们需要查询的内容。 索引中记录了表中的列值（一列或多列）与其物理位置之间的对应关系。

索引从本质上来说就是一些数据的键与元组标识符（TID）之间的映射，这些标识符确定了该索引键值在表中对应的元组。

在PostgreSQL中，索引是一种特殊的数据库对象，主要用于加速数据访问。索引是辅助结构：可以从表中的信息中删除和重建每个索引。索引还可用于强制执行某些完整性约束，如唯一索引还可以起到唯一约束的作用。

尽管索引类型（也称为访问方法）之间存在差异，但它们最终都将一个键(例如，索引列的值)与包含该键的表的行关联起来。每一行由TID（元组id）标识，它由文件中的块数和块内行的位置组成。也就是说，使用索引访问，我们可以快速读取那些包含我们需要的信息的行，而无需扫描整个表。

 维护索引也需要一定的维护成本。对于表上数据的每个操作，无论是表行的插入，删除还是更新，都需要在同一个事务中更新该表的索引。注意：在块内更新表字段不会导致索引更新，这种技术叫做HOT（Heap-OnlyTuples）。

建立索引的好处是加快对表中记录的查找或排序。 但建索引要付出以下代价：

1.增加了数据库的存储空间。

2.在插入和修改数据时要花费较多的时间， 因为索引也要随之更新。

索引自身也占用存储空间、消耗计算资源，创建过多的索引将对数据库性能造成负面影响。因此，仅在必要时创建索引。索引与它的表文件是的分离。从本质来讲，索引是数据的键和元组标识符（TID）之间的映射。

**索引的作用**

**1.加速条件检索**： 利用索引快速定位到可能满足条件的记录，不需要遍历所有记录。

explain analyze select \* from t1 where id = 10000;--无索引时的查询计划（执行时间较长)

explain analyze select \* from t1 where id = 10000;--添加索引后的查询计划(执行时间大大缩短)

**2.加速排序**：

利用索引可直接返回升序和降序的排序结果，无需在每次执行查询时再排序。

explain analyze select \* from t1 order by id desc;--不利用索引排序时的执行计划（执行时间较长)

explain analyze select \* from t1 order by id desc;--利用索引排序时的执行计划(执行时间大大缩短)

只返回TopN排序结果的查询，索引的优势更明显。

explain analyze select \* from t1 order by id desc limit 10;--不利用索引排序时的执行计划（执行时间较长）

explain analyze select \* from t1 order by id desc limit 10;--利用索引排序时的执行计划 (执行时间大大缩短)

**3.加速Join**

●索引可加速嵌套循环Join和合并Join。

嵌套循环Join利用索引能快速检索到内层表的关联记录

创建一个连接查询用的小表t2

create table t2(like t1);

insert into t2 select \* from t1 limit 10;

create index on t2(id);

explain analyze select \* from t1,t2 where t1.id=t2.id;--执行tb1和tb2的Join查询

归并Join利用索引能快速获得排序后的结果

创建一个连接查询用的较大的表t3

create table t3(like t1);

insert into t3 select \* from t1 limit 1000;

create index on t3(id);

explain analyze select \* from t1,t3 where t1.id=t3.id;--执行tb1和tb3的Join查询

●在没有索引时，优化器更倾向于使用HashJoin,但通常比有索引时的成本更高。

临时禁用索引扫描

set enable\_indexscan=off;

set enable\_bitmapscan=off;

explain analyze select \* from t1,t3 where t1.id=t3.id;--执行t1和t3的Join查询

**索引的代价**

**1.影响更新速度**

●对表记录的增，删，改需要更新相应的索引

●对频繁更新的表设置较小的fillfactor可降低索引更新的需求(HOT更新)

●使用下面的SQL可检查HOT更新比例

select schemaname,relname,n\_tup\_upd, n\_tup\_hot\_upd from pg\_stat\_user\_tables;

**2.空间占用**

●可通过pg\_indexes\_size()和pg\_relation\_size()查看索引大小

select pg\_indexes\_size('t1');
select pg\_relation\_size('idx\_t1\_id');

**3.索引的利用率**

●通过pg\_stat\_user\_indexes. idx\_scan可检查利用索引进行扫描的次数

select \* from pg\_stat\_user\_indexes;

仅创建必要的索引！

**索引类型**

pg\_am （“am”代表访问方法）存储关于关系访问方法的信息。

SELECT amname FROM pg\_am where amtype='i';

amname

btree

hash

gist

gin

spgist

brin

(6 rows)

每种索引类型使用不同的存储结构和算法来处理不同种类的查询。

**索引属性**

属性使用特殊函数进行查询，并被分为多个层次：

（1）访问方法属性 - pg\_indexam\_has\_property

（2）特定索引的属性- pg\_index\_has\_property

（3）索引各列的属性- pg\_index\_column\_has\_property

**1.访问方法的属性（通过 btree的示例）**

select

a.amname,

p.name,

pg\_indexam\_has\_property(a.oid,p.name)

from

pg\_am a,unnest(array['can\_order','can\_unique','can\_multi\_col','can\_exclude']) p(name)

where a.amname = 'btree' order by a.amname;

（1） can\_order：创建索引时是否可以指定值的排列顺序（仅适用于«btree»）。

（2）  can\_unique：是否支持唯一约束和主键约束（仅适用于«btree»）。

（3） can\_multi\_col：是否可以在多个列上创建索引。

（4） can\_exclude：是否支持排排他束EXCLUDE。

**2.特定索引的属性（以一个现有的索引为例）**

select p.name, pg\_index\_has\_property('t1\_age\_idx'::regclass,p.name) from unnest(array['clusterable','index\_scan','bitmap\_scan','backward\_scan'])p(name);

（1） clusterable：是否可以根据索引重新聚集（使用同名命令CLUSTER进行）。

（2） index\_scan：是否支持索引扫描。

（3） bitmap\_scan ：是否支持位图扫描。

（4） backward\_scan ：是否可以按创建索引的相反顺序返回结果

**2.索引各列的属性**

select

p.name,

pg\_index\_column\_has\_property('t1\_id\_idx'::regclass,1,p.name)

from

unnest(array['asc','desc','nulls\_first','nulls\_last','orderable','distance\_orderable','returnable','search\_array','search\_nulls']) p(name);

（1） asc，desc，nulls\_first，nulls\_last，orderable：这些属性与值的排序有关。

（2） distance\_orderable：结果是否可以按照一个"distance"操作符 （例如ORDER BY col <-> constant）确定的排列顺序返回（目前仅适用于GiST和RUM索引）。

（3） returnable：是否可以在不访问表的情况下使用索引，即支持仅索引扫描。

（4） search\_array：是否支持使用表达式：

indexed-field IN（list\_of\_constants）搜索多个值，

该表达式

与 indexed-field=ANY（array\_of\_constants）相同。

（5）search\_nulls

是否可以通过IS NULL和IS NOT NULL条件进行搜索。

**索引操作符类**

创建索引时可以为索引中的每一列都指定一个操作符类。操作符类定义索引列上能够使用什么操作符。例如，在int4的数据类型上创建一个B树索引，默认会使用int4\_ops类，这个操作符类包括用于int4类型值的比较函数。实际上，在大部分场景中，用默认操作符类就足够了。

除了默认的操作符类，还有一些内建的操作符类：

操作符类text\_pattern\_ops、varchar\_pattern\_ops和 bpchar\_pattern\_ops分别支持类型text、varchar和 char上的B树索引。它们与默认操作符类的区别是值的比较是严格按照字符来进行排序，而不是根据地区的排序规则来排序。它们适用于非 C 区域中的模式匹配（LIKE或正则表达式）。

示例：

CREATE INDEX idx\_t1\_name ON t1 (name text\_pattern\_ops);

注意：如果涉及 <、<=、>、>= 等比较查询，应该创建默认操作符类的索引。这些查询不能使用xxx\_pattern\_ops操作符类。如果使用C区域，也不需要xxx\_pattern\_ops操作符类，因为在C区域中的模式匹配查询可以用带有默认操作符类的索引。

下面的查询展示了所有已定义的操作符类：

SELECT

am.amname AS index\_method,

opc.opcname AS opclass\_name,

opc.opcintype::regtype AS indexed\_type,

opc.opcdefault AS is\_default

FROM pg\_am am, pg\_opclass opc

WHERE opc.opcmethod = am.oid

ORDER BY index\_method, opclass\_name;

在大多数情况下，我们不需要了解有关运算符类的任何信息。通常我们只是创建一个索引，默认情况下使用某个运算符类。

但是，我们可以显式指定运算符类。这是一个简单的示例，：在排序规则为非C的数据库中，常规索引不支持LIKE操作：

show lc\_collate;

explain  select \* from t1 where name like ‘xxx%';  --Seq Scan on

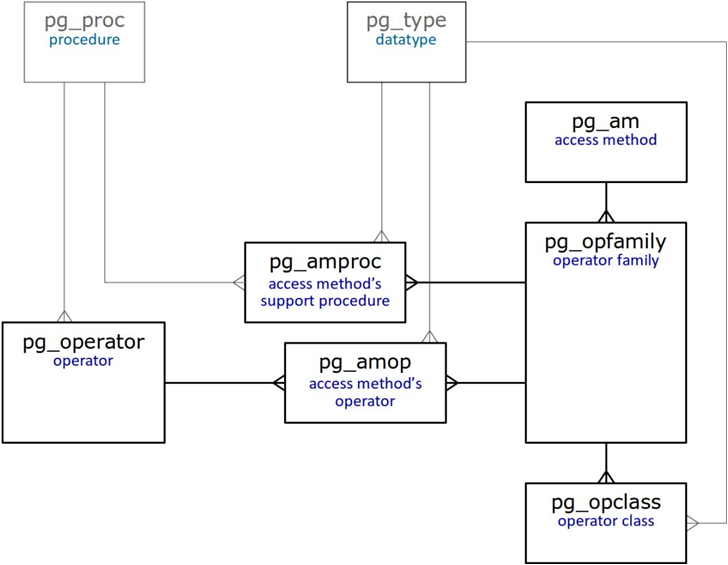
我们可以使用操作符类 text\_pattern\_ops 来克服此限制：

create index on t1(name text\_pattern\_ops);

explain select \* from t1 where name like ‘xxx%';  --BitmapIndex Scan on

**系统目录**

下图是系统目录中与运算符类和族直接相关的表的简化图。



系统目录使我们无需查找文档即可找到许多问题的答案。

例如，索引支持哪些操作符类？

select

opcname, opcintype::regtype

from

pg\_opclass

where opcmethod = (select oid from pg\_am where amname = 'btree')

order by opcintype::regtype::text;

操作符类包含操作符（也就是说，索引可以使用哪些操作符）？

select

amop.amopopr::regoperator

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 = 'btree' and amop.amoplefttype = opc.opcintype;