**表的存储属性**

**概述**

TOAST（The Oversized-Attribute Storage Technique），超尺寸属性存储技术，又称行外存储技术。

在PG中，页是数据在文件存储中的基本单位，默认的大小为8 KB。同时，PG不允许元组（即一行数据）跨页存储，那么对于超长数据，PG 就会启动 TOAST ，具体就是采用压缩和切片的方式。

TOAST的主要作用：存储大字段的值。

TOAST表不能单独创建，如果一张表中存在可变长数据类型，并且表包含了main,extended或external存储格式的字段时，系统会自动创建一个和普通表相关联且唯一的TOAST表。当数据超过TOAST\_TUPLE\_THRESHOLD(默认2KB)时，就会压缩数据，如果压缩完的数据也>2KB，则将会被拆分为更小的块，并存储在相关TOAST表中的多个物理行（行外存储）。每个原始字段值都被一个指针（chunk\_id）替换，根据这个指针可以找到行外存储的数据。

注意：这一技术对用户来说是完全透明的，用户不需要关注这一技术实现。 TOAST表的OID存储在pg\_class.reltoastrelid字段中， TOAST表由 3 列组成：

chunk\_id：堆中引用的块的编号

chunk\_seq：chunk 的序列号，用来表示该行数据在整个数据中的位置。同一个chunk\_id如果大于2KB，将被切片存储。这里存储切片后的序号。

chunk\_data :实际存储的数据

字段有4种TOAST策略：

1.PLAIN ：避免压缩和行外存储。只有那些不需要 TOAST 策略就能存放的数据类型的默认策略，比如整数类型(INT，SMALLINT，BIGINT)、布尔类型(BOOLEAN)，而对于 text 这类要求存储长度超过页大小的类型，是不允许采用此策略的。

2.EXTENDED ：允许压缩和行外存储。大多数可以使用TOAST机制的数据类型默认存储策略。一般会先压缩，如果还是太大，就会行外存储

3.EXTERNAL ：允许行外存储，但不许压缩。类似字符串这种会对数据的一部分进行操作的字段，采用此策略可能获得更高的性能，因为不需要读取出整行数据再解压。代价是牺牲存储空间。

4.MAIN ：允许压缩，但不许行外存储。不过实际上，为了保证过大数据的存储，行外存储在其它方式（例如压缩）都无法满足需求的情况下，作为最后手段还是会被启动。因此理解为：尽量不使用行外存储更贴切。是numeric是默认存储策略。

\d+ tablename 查看表的toast属性

select relname from pg\_class where oid = (select reltoastrelid from pg\_class where relname='t1');--查看与原始表相关的toast表

通常表名为：pg\_toast\_(OID)，其中OID是toast表的OID，即原始表的reltoastrelid。

ALTER TABLE t1 ALTER name SET STORAGE EXTERNAL; --更改toast测试

\d pg\_toast.pg\_toast\_24672--查看toast字段

栏位 | 类型

------------+---------

chunk\_id | oid

chunk\_seq | integer

chunk\_data | bytea

**测试**

场景一：TOAST表只在有变长字段，并且存储为main,extended或external时才会创建。

-- 1. 存储格式是 main 会创建 toast 表

create table t1 (id numeric);

select relname,reltoastrelid from pg\_class where relname='t1';

-- 2. 存储格式是 plain 不会创建 toast 表

create table t2 (id int);

select relname,reltoastrelid from pg\_class where relname='t2';

场景二:

insert into t1 values(1, 'abcdefghij');

select \* from pg\_toast. pg\_toast\_24854;--没数据

因为 name 只有10个字符，所以没有压缩，也没有行外存储。然后我们增加 name 的长度，看看会发生什么情况？

update t1 set name=repeat('abcdefghij',10000) where id=1;

select id,length(name) from t1;

id | length

----+--------

1 | 100000

select \* from pg\_toast. pg\_toast\_24854; --没数据

增加name字段长度，直到pg\_toast. pg\_toast\_24854表中有数据：

update t1 set name=repeat('abcdefghij',20000) where id=1;

select id,length(name) from t1;

id | length

----+--------

1 | 200000

select chunk\_id,chunk\_seq,length(chunk\_data) from pg\_toast. pg\_toast\_24854; chunk\_id | chunk\_seq | length

----------+-----------+--------

25155 | 0 | 1996

25155 | 1 | 310

可以看到，直到 name 的长度为200000时（已远远超过页大小 8K），对应 TOAST 表中才有了2行数据，且长度都是略小于2K，这是因为 extended 策略下，先启用了压缩，然后才使用行外存储。

场景三：

将 name 的 TOAST 策略改为 EXTERNA ，以禁止压缩。

alter table t1 alter name set storage external;

\d+ t1

然后我们再插入一条数据：

insert into t1 values(2, 'abcdefghij'); select id,length(name) from t1;

id | length

----+--------

1 | 200000

2 | 10

重复以上步骤，直到TOAST表中产生新的行：

update t1 set name=repeat('abcdefghij',205) where id=2;

select id,length(name) from t1;

id | length

----+--------

1 | 200000

2 | 2050

select chunk\_id,chunk\_seq,length(chunk\_data) from pg\_toast. pg\_toast\_24854;

chunk\_id | chunk\_seq | length

----------+-----------+--------

25155 | 0 | 1996

25155 | 1 | 310

25157 | 0 | 1996

25157 | 1 | 54

这次当 name 长度达到2050（按照官方文档，应该是超过2KB左右）， TOAST 表中产生了新的2条chunk\_id 为25157的行，且2行数据的 chunk\_data 的长度之和正好等于2050。

总结：

如果策略允许压缩，则TOAST优先选择压缩。

不管是否压缩，一旦数据超过2KB左右，就会启用行外存储。

修改TOAST策略，不会影响现有数据的存储方式

**临时表**

PostgreSQL支持两种临时表：

1.会话级的临时表

2.事务级的临时表

在会话级别的临时表中， 数据可以一直保存在整个会话的生命周期中， 而在事务级别的临时表中， 数据只存在于这个事务的生命周期中。默认创建的是会话级别的临时表。不管是会话级还是事务级的临时表，当会话结束后，临时表会消失。

create [global|local] TEMPORARY table tmp1(id int, name text);

create TEMP table tmp1(id int, name text);

注意：临时表存放在一个特殊的Schema下， Schema名为"pg\_temp\_xx"， 其中的"xx"代表一个数字， 如"2""3"等， 但不同的session数字是不同的。

为了与其他数据库兼容， 还提供了 GLOBAL 和 LOCAL 两个关键字， 但这两个关键字没有任何作用。

另打开一个psql， 查看当前的表，是看不到这张表的。

create TEMPORARY table tmp2(id int primary key, info text) on commit delete rows; --创建事务级的表

事务一旦结束， 这种临时表中的数据就会消失。

ON COMMIT有3种形式:

1.ON COMMIT PRESERVE ROWS： 若不带"ON COMMIT"子句， 默认情况下， 数据会一直存在于整个会话周期中。

2.ON COMMIT DELETE ROWS： 数据只存在于事务周期中， 事务提交后数据就消失了。

3.ON COMMIT DROP： 数据只存在于事务周期中， 事务提交后临时表就消失了。 这种情况下， 创建临时表的语句与插入数据的语句需要放到一个事务中， 若把创建临时表的语句放在一个单独的事务中， 事务一旦结束， 这张临时表就会消失。

**UNLOGGED表**

UNLOGGED表是是通过禁止产生WAL日志的方式提升写性能。 因为没有WAL日志， 所以表的内容无法在主备库直接同步， 如果此时数据库异常宕机， 表的内容将丢失， 所以可以把UNLOGGED表称为"半临时表"。当然如果数据库是正常关机的， 则UNLOGGED表的内容不会丢失。

UNLOGGED表在使用上与普通表没有区别， 仅仅在插入、 删除、 更新数据时不产生WAL日志， 所以做这些DML操作的性能会更高。 另外需要注意的是， 数据库异常宕机时， UNLOGGED表的数据可能会丢失。

CREATE UNLOGGED TABLE u1(id int, name text);

select pg\_walfile\_name(pg\_current\_wal\_lsn());

pg\_waldump -f walifle

walminer 日志解析工具

1.yum -y install git

2.git clone https://gitee.com/movead/XLogMiner.git

3.将walminer目录放置到"../contrib/"目录下

4.进入walminer目录

5.执行命令

make && make install

6.创建walminer的extension

create extension walminer;

7.添加要解析的wal日志文件

select walminer\_wal\_add('/home/postgres/pgdata/pg\_wal');

8.Remove wal日志文件

select walminer\_wal\_remove('/home/postgres/pgdata/pg\_wal');

9.List wal日志文件

select walminer\_wal\_list();

10.执行解析

​ select walminer\_all();

或 select wal2sql();

11.解析结果查看

select \* from walminer\_contents;

**表继承**

表继承是PostgreSQL特有的，子表可以从父表继承字段和一些属性。

所有父表的检查约束和非空约束都会自动被所有子表继承。 不过其他类型的约束（唯一、 主键、 外键） 不会被继承。

一个子表可以从多个父表中继承，子表将拥有所有父表的列和自己定义的列。如果同一个字段名出现在多个父表中，或者同时出现在父表和子表的定义中，那么这些字段将会被"融合"，因此在子表中就只有一个这样的字段。但是这种"融合"要求字段的数据类型相同，否则会报错。融合的字段拥有父表字段的所有检查约束。

采用select、update、delete等命令操作父表时，也会同时操作相应的子表，当使用alter table修改父表的表结构时，也会同时修改子表的表结构，但"reindex"、"vacuum"命令不会影响到子表。此外，唯一约束、外键的作用域也不会扩大到子表上。

创建一张 person 作为父表：

CREATE TABLE person (id int primary key,name text not null,age int check(age>0));

创建子表 student，student 表比 person 表多一个字段 class\_id：

CREATE TABLE student (class\_id int) INHERITS (person);

查看 person、student 的表结果：

\d person

\d student

向子表 students 表中插入两条记录，插入成功后，在父表也可以看到这两条记录：

insert into student values(1,'zhangsan',15,1);

insert into student values(2,'lisi',14,2);

select \* from person;

select \* from student;

修改子表 students 中的数据，在父表中也可以查到修改后的结果：

update student set age=13 where name='zhangsan';

select \* from persons;

但向父表 person 中插入一条数据，在子表 student 中却查不到这条数据：

insert into person values(3,'wangwu',18);

select \* from person;

select \* from student;

查看数据来自于哪张表

select tableoid,\* from person;

select tableoid::regclass,\* from person;

总结:当查询父表时，会把子表中的数据也查询出来，反之则不行。如果只想查询父表自己的数据，在表名前加"only"即可：

select \* from only person;