《数据库系统原理》课程实验指导

openGauss 数据库物理设计

2023 年 9 月

目录

前 言

实验环境说明

本实验环境为 virtualBOX 虚拟机 openEuler20.03 系统上的 openGauss1.1.0/openGauss2.0.0 数据库和华为云 GaussDB(openGauss)数据库。

特别说明:

华为云数据库 GaussDB(openGauss)是商业版的，权限管控是比较严格的。由于创建和管理表空间的命令需要 sysadmin 权限，这个权限是顶级的，无法授予做实验的学生。因此，表空间实验无法在华为云数据库中做，建议采用本机安装的 openGauss 作为实验平台。

1 表空间

1.1 实验介绍

关于本实验

本实验主要描述 openGauss 数据库如何创建和管理表空间

实验目的

1.学会创建多个表空间，并在不同的表空间上创建对象

2.对表空间进行查询，删除等管理操作

特别说明:

华为云数据库 GaussDB(openGauss)是商业版的，权限管控是比较严格的。由于创建和管理表空间的命令需要 sysadmin 权限，这个权限是顶级的，无法授予做实验的学生。因此，表空间实验无法在华为云数据库中做，建议采用本机安装的 openGauss 作为实验平台。

1.2 表空间的应用背景

通过使用表空间，管理员可以控制一个数据库安装的磁盘布局。这样有以下特点:
1.如果初始化数据库所在的分区或者卷空间已满，又不能逻辑上扩展更多空间，可以在不同的分区上创建和使用表空间，直到系统重新配置空间。
2.表空间允许管理员根据数据库对象的使用模式安排数据位置，从而提高性能。
3.一个频繁使用的索引可以放在性能稳定且运算速度较快的磁盘上，比如一种固态设备。
4.一个存储归档的数据，很少使用的或者对性能要求不高的表可以存储在一个运算速度较慢的磁盘上。
5.管理员通过表空间可以设置占用的磁盘空间。用以在和其他数据共用分区的时候，防止表空间占用相同分区上的其他空间。
6.表空间可以控制数据库数据占用的磁盘空间，当表空间所在磁盘的使用率达到 90%时，数据库将被设置为只读模式，当磁盘使用率降到 90%以下时，数据库将恢复到读写模式。
7.建议用户使用数据库时，通过后台监控程序或者 Database Manager 进行磁盘空间使用率监控，以免出现数据库只读情况。
8.表空间对应于一个文件系统目录，比如: '数据库节点数据目录/pg_location/tablespace/tablespace_1'是用户拥有读写权限的空目录。
9.使用表空间配额管理会使性能有 30%左右的影响，MAXSIZE 指定每个数据库节点的配额大小，误差范围在 500MB 以内。请根据实际的情况确认是否需要设置表空间的最大值。

1.3 创建表空间

以操作系统用户 omm 登录数据库主节点。

```
su - omm
```

启动 openGauss 数据库服务

```
gs_om -t start
```

使用如下命令连接数据库。

```
gsql -d tpch -p 26000 -r
```

创建用户 jack

```
CREATE USER jack IDENTIFIED BY ‘openeuler12345!’;
```

创建成功

```
tpch=# create user jack identified by 'openeuler12345!';
CREATE ROLE
tpch=#
```

创建表空间

```
CREATE TABLESPACE fastspace RELATIVE LOCATION ‘tablespace/tablespace_1’;
```

其中“fastspace”为新创建的表空间，“$dataNode/pg_location/tablespace/tablespace_1_”是用户拥有读写权限的空目录。

创建成功

```
tpch=# create tablespace fastspace relative location 'tablespace/tablespace_1';
CREATE TABLESPACE
```

数据库系统管理员（本例中，为 omm 用户）执行如下命令将“fastspace”表空间的访问权限赋予数据用户 jack。

```
GRANT CREATE ON TABLESPACE fastspace TO jack;
```

赋予成功

```
tpch=# grant create on tablespace fastspace to jack;
GRANT
tpch=#
```

同理，可以创建多个表空间

```
tpch=# create tablespace example2 relative location 'tablespace/tablespace_2';
CREATE TABLESPACE
tpch=# create tablespace example3 relative location 'tablespace/tablespace_3';
CREATE TABLESPACE
tpch=# create tablespace example4 relative location 'tablespace/tablespace_4';
CREATE TABLESPACE
tpch=#
```

1.4 在表空间上创建对象

如果用户拥有表空间的 CREATE 权限，就可以在表空间上创建数据库对象。操作系统管理员（omm 用户）具有以上表空间的 CREATE 权限，并且 jack 用户拥有 fastspace 表空间的 CREATE 权限。
在指定的表空间上创建表（创建其他的对象方法类似）：

```
CREATE TABLE table_1(i int) TABLESPACE fastspace;
CREATE TABLE table_2(i int) TABLESPACE example2;
```

创建成功

```
tpch=# create table table_1(i int) tablespace fastspace;
CREATE TABLE
tpch=# create table table_2(i int) tablespace example2;
CREATE TABLE
```

在默认表空间上创建表：
先使用 set default_tablespace 设置默认表空间

```
SET default_tablespace = ‘example3’;
```

```
tpch=# set default_tablespace='example3';
SET
```

再创建表，这样无需指定表空间，表创建在默认表空间

```
CREATE TABLE table_3(i int);
```

```
tpch=# create table table_3(i int);
CREATE TABLE
```

1.5 管理表空间

1.5.1 查询表空间

方式 1：检查 pg_tablespace 系统表。如下命令可查到系统和用户定义的全部表空间。

```
tpch=# SELECT spcname FROM pg_tablespace;
```

```
tpch=# select spcname from pg_tablespace;
  spcname
------------
 pg_default
 pg_global
 fastspace
 example2
 example3
 example4
(6 rows)
```

方式 2：使用 gsql 程序的元命令查询表空间。

```
tpch=# \db
```

```
tpch=# \db
             List of tablespaces
    Name    | Owner |        Location
------------+-------+-------------------------
 example2   | omm   | tablespace/tablespace_2
 example3   | omm   | tablespace/tablespace_3
 example4   | omm   | tablespace/tablespace_4
 fastspace  | omm   | tablespace/tablespace_1
 pg_default | omm   |
 pg_global  | omm   |
(6 rows)
```

1.5.2 查询表空间当前使用情况

使用 PG_TABLESPACE_SIZE(‘表空间名字’)来查询

```
tpch=# SELECT PG_TABLESPACE_SIZE(‘fastspace’);
```

```
tpch=# select pg_tablespace_size('fastspace');
 pg_tablespace_size
--------------------
               8192
(1 row)
```

其中 8192 表示表空间的大小 ， 单位是字节。

1.5.3 重命名表空间

执行如下命令对表空间 fastspace 重命名为 example。

```
tpch=# ALTER TABLESPACE fastspace RENAME TO example;
```

改名成功

```
tpch=# alter tablespace fastspace rename to example;
ALTER TABLESPACE
tpch=# \db
             List of tablespaces
    Name    | Owner |        Location
------------+-------+-------------------------
 example    | omm   | tablespace/tablespace_1
 example2   | omm   | tablespace/tablespace_2
 example3   | omm   | tablespace/tablespace_3
 example4   | omm   | tablespace/tablespace_4
 pg_default | omm   |
 pg_global  | omm   |
(6 rows)
```

1.5.4 删除表空间

```
tpch=# DROP TABLESPACE example;
```

说明：用户必须是表空间的 owner 或者系统管理员才能删除表空间。

```
tpch=# drop tablespace example;
ERROR:  tablespace "example" is not empty
```

删除失败，表空间不为空的情况下无法删除表空间(避免误删里面的重要数据)。
先清空表空间

```
tpch=# DROP TABLE table_1;
```

删除成功

```
tpch=# drop table table_1;
DROP TABLE
```

然后在删除表空间

```
tpch=# drop tablespace example;
DROP TABLESPACE
```

删除成功

```
tpch=# \db
             List of tablespaces
    Name    | Owner |         Location
------------+-------+--------------------------
 example2   | omm   | tablespace/tablespace_2
 example3   | omm   | tablespace/tablespace_3
 example4   | omm   | tablespace/tablespace_4
 pg_default | omm   |
 pg_global  | omm   |
(5 rows)
```

2 分区表

2.1 实验介绍

关于本实验

本实验主要描述 openGauss 数据库如何创建分区表，插入数据以及对分区表进行管理

实验目的

1.学会创建分区表，并向表中插入数据，进行观察

2.对分区表进行重命名，删除等管理操作

2.2 分区表应用背景

openGauss 是基于 PostgreSQL9.2.4 的内核开发的，在 PostgreSQL10 之前要达到实现分区表的效果可以有两种方式，一种是使用继承的触发器函数来实现，一种是安装 pg_pathman 的插件来实现，直到 PostgreSQL10 才引入了 partition 的语法；但是 openGauss 从开源发布就可以直接使用 partition 的方式来创建分区表，行存表支持范围分区和间隔分区，列存表支持范围分区。
openGauss 数据库支持的分区表为范围分区表、列表分区表、哈希分区表。
范围分区表：将数据基于范围映射到每一个分区，这个范围是由创建分区表时指定的分区键决定的。这种分区方式是最为常用的，并且分区键经常采用日期，例如将销售数据按照月份进行分区。
列表分区表：将数据中包含的键值分别存储再不同的分区中，依次将数据映射到每一个分区，分区中包含的键值由创建分区表时指定。
哈希分区表：将数据根据内部哈希算法依次映射到每一个分区中，包含的分区个数由创建分区表时指定。
分区表和普通表相比具有以下优点：
改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索效率。
增强可用性：如果分区表的某个分区出现故障，表在其他分区的数据仍然可用。
方便维护：如果分区表的某个分区出现故障，需要修复数据，只修复该分区即可。
均衡 I/O：可以把不同的分区映射到不同的磁盘以平衡 I/O，改善整个系统性能。
普通表若要转成分区表，需要新建分区表，然后把普通表中的数据导入到新建的分区表中。因此在初始设计表时，请根据业务提前规划是否使用分区表。

openGauss分区表限制和特点：

1、主键约束或唯一约束必须要包含分区字段
2、分区表表名只能在 pg_partition 视图中查看，在 pg_tables 和 pg_stat_all_tables 中无法查到
3、分区表索引在 opengauss 里分 local 和 global，默认是 global
4、分区个数不能超过 327675、

5、选择分区使用 PARTITION FOR()，括号里指定值个数应该与定义分区时使用的列个数相同，并且一一对应。
6、Value 分区表不支持相应的 Alter Partition 操作
7、列存分区表不支持切割分区
8、间隔分区表不支持添加分区

2.3 创建分区表

2.3.1 方法一：VALUES LESS THAN

语法：PARTITION BY RANGE(partition_key)**从句是** VALUES LESS THAN **的语法格式，范围分区策略的分区键最多支持** 4 **列。**

PARTITION partition_name VALUES LESS THAN ({ partition_value | MAXVALUE })

· **每个分区都需要指定一个上边界。**

· **分区上边界的类型应当和分区键的类型一致。**
· 分区列表是按照分区上边界升序排列的，值较小的分区位于值较大的分区之前。
实例：

```
create table partition_orders_1(
o_orderkey integer,
o_custkey integer,
o_orderstatus char(1),
o_totalprice decimal(15,2),
o_orderdate date,
o_orderpriority char(15),
o_clerk char(15),
o_shippriority integer,
o_comment varchar(79),
PRIMARY KEY (o_orderkey)
)
partition by range(o_orderkey)
(
partition p1 values less than(100),
partition p2 values less than(200),
partition p3 values less than(300),
partition p4 values less than(maxvalue)
);
```

此分区表分区键为 id，分了 4 个区，分别是 p1<100, 100<=p2<200, 200<=p3<300, 300<=p4。

```
[omm@db1 openGauss]$ gsql -d tpch -p 26000 -f task2.sql
gsql:task2.sql:19: NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit inde
x "partition_orders_1_pkey" for table "partition_orders_1"
CREATE TABLE
total time: 40  ms
```

执行\d+ 命令显示该分区表的详细信息

```
tpch=# \d+ partition_orders_1
                                  Table "public.partition_orders_1"
     Column      |              Type              | Modifiers | Storage  | Stats
 target | Description
----------------+--------------------------------+-----------+----------+------
--------+-------------
 o_orderkey      | integer                        | not null  | plain    |
        |
 o_custkey       | integer                        |           | plain    |
        |
 o_orderstatus   | character(1)                   |           | extended |
        |
 o_totalprice    | numeric(15,2)                  |           | main     |
        |
 o_orderdate     | timestamp(0) without time zone |           | plain    |
        |
 o_orderpriority | character(15)                  |           | extended |
        |
 o_clerk         | character(15)                  |           | extended |
        |
 o_shippriority  | integer                        |           | plain    |
        |
 o_comment       | character varying(79)          |           | extended |
        |
Indexes:
    "partition_orders_1_pkey" PRIMARY KEY, btree (o_orderkey) LOCAL(PARTITION p1
_o_orderkey_idx, PARTITION p2_o_orderkey_idx, PARTITION p3_o_orderkey_idx, PARTI
TION p4_o_orderkey_idx)  TABLESPACE pg_default
Range partition by(o_orderkey)
Number of partition: 4 (View pg_partition to check each partition range.)
Has OIDs: no
Options: orientation=row, compression=no
```

通过 select 命令列出各分区情况

```
tpch=# select relname,parttype,parentid,boundaries from pg_partition where parentid in(select oid from pg_class
where relname=' partition_orders_1' );
```

```
tpch=# select relname,parttype,parentid,boundaries from pg_partition where parentid i
n(select oid from pg_class where relname='partition_orders_1');
      relname       | parttype | parentid | boundaries
--------------------+----------+----------+------------
 partition_orders_1 | r        |    16763 |
 p1                 | p        |    16763 | {100}
 p2                 | p        |    16763 | {200}
 p3                 | p        |    16763 | {300}
 p4                 | p        |    16763 | {NULL}
(5 rows)
```

2.3.2 方法二：START END

语法：PARTITION BY RANGE(partition_key)从句是 START END 的语法格式，范围分区策略的分区键仅支持 1 列。

PARTITION partition_name {START(partition_value) END(partition_value) EVERY(interval_value)} | {START(partition_value) END(partition_value | MAXVALUE)} | {START(partition_value)} | {END(partition_value | MAXVALUE)}

· 在创建分区表若第一个分区定义含 START 值，则范围（MINVALUE，START）将自动作为实际的第一个分区。

· 每个 partition_start_end_item 中的 START 值（如果有的话，下同）必须小于其 END 值；

· 相邻的两个 partition_start_end_item，第一个的 END 值必须等于第二个的 START 值；

· 每个 partition_start_end_item 中的 EVERY 值必须是正向递增的，且必须小于（END-START）值；

· 每个分区包含起始值，不包含终点值，即形如：[起始值，终点值)，起始值是 MINVALUE 时则不包含；

· 一个 partition_start_end_item 创建的每个分区所属的 TABLESPACE 一样；

· partition_name 作为分区名称前缀时，其长度不要超过 57 字节，超过时自动截断；

· 在创建、修改分区表时请注意分区表的分区总数不可超过最大限制（32767）；

· 在创建分区表时 START END 与 LESS THAN 语法不可混合使用。
· 即使创建分区表时使用 START END 语法，备份（gs_dump）出的 SQL 语句也是 VALUES LESS THAN
· 单一 start 分区不能紧挨着单一 end 分区，否则会报错
实例：

```
create table partition_orders_2(
o_orderkey integer,
o_custkey integer,
o_orderstatus char(1),
o_totalprice decimal(15,2),
o_orderdate date,
o_orderpriority char(15),
o_clerk char(15),
o_shippriority integer,
o_comment varchar(79),
PRIMARY KEY (o_orderkey)
)
partition by range(o_orderkey)
(
partition p1 start(2) end(100) every(10),
partition p2 end(200),
partition p3 end(300),
partition p4 start(300),
partition p5 start(400),
partition p6 start(500) end(600)
);
```

此实例第一个分区定义含 start，则范围（minvalue，2）自动作为第一个分区 p1_0,p1_0<2
由于 every（10），则 p1 分区进行间隔分区，间隔为 10
即 p1_1 [2,12),p1_2 [12,22),p1_3 [22,32),p1_4 [32,42),p1_5 [42,52),p1_6 [52,62),p1_7 [62,72) p1_8 [72,82),p1_9 [82,92),p1_10 [92,100)
之后 5 个分区，p2 [100,200),p3 [200,300),p4 [300,400),p5 [400,500),p6 [500,600)

```
[omm@db1 openGauss]$ gsql -d tpch -p 26000 -f task2.sql
gsql:task2.sql:21: NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit inde
x "partition_orders_2_pkey" for table "partition_orders_2"
CREATE TABLE
total time: 42  ms
```

执行\d+ 命令显示该分区表的详细信息

```
tpch=# \d+ partition_orders_2
                                  Table "public.partition_orders_2"
     Column      |              Type              | Modifiers | Storage  | Stats
 target | Description
-----------------+--------------------------------+-----------+----------+------
--------+-------------
 o_orderkey      | integer                        | not null  | plain    |
        |
 o_custkey       | integer                        |           | plain    |
        |
 o_orderstatus   | character(1)                   |           | extended |
        |
 o_totalprice    | numeric(15,2)                  |           | main     |
        |
 o_orderdate     | timestamp(0) without time zone |           | plain    |
        |
 o_orderpriority | character(15)                  |           | extended |
        |
 o_clerk         | character(15)                  |           | extended |
        |
 o_shippriority  | integer                        |           | plain    |
        |
 o_comment       | character varying(79)          |           | extended |
        |
Indexes:
    "partition_orders_2_pkey" PRIMARY KEY, btree (o_orderkey) LOCAL(PARTITION p1
_0_o_orderkey_idx, PARTITION p1_1_o_orderkey_idx, PARTITION p1_2_o_orderkey_idx,
 PARTITION p1_3_o_orderkey_idx, PARTITION p1_4_o_orderkey_idx, PARTITION p1_5_o_
orderkey_idx, PARTITION p1_6_o_orderkey_idx, PARTITION p1_7_o_orderkey_idx, PART
ITION p1_8_o_orderkey_idx, PARTITION p1_9_o_orderkey_idx, PARTITION p1_10_o_orde
rkey_idx, PARTITION p2_o_orderkey_idx, PARTITION p3_o_orderkey_idx, PARTITION p4
_o_orderkey_idx, PARTITION p5_o_orderkey_idx, PARTITION p6_o_orderkey_idx)  TABL
ESPACE pg_default
Range partition by(o_orderkey)
Number of partition: 16 (View pg_partition to check each partition range.)
Has OIDs: no
Options: orientation=row, compression=no
```

通过 select 命令列出各分区情况

```
tpch=# select relname,parttype,parentid,boundaries from pg_partition where parentid in(select oid from pg_class
where relname=' partition_orders_2' );
```

```
tpch=# select relname,parttype,parentid,boundaries from pg_partition where paren
tid in(select oid from pg_class where relname='partition_orders_2');
      relname       | parttype | parentid | boundaries
--------------------+----------+----------+------------
 partition_orders_2 | r        |    16777 |
 p1_0               | p        |    16777 | {2}
 p1_1               | p        |    16777 | {12}
 p1_2               | p        |    16777 | {22}
 p1_3               | p        |    16777 | {32}
 p1_4               | p        |    16777 | {42}
 p1_5               | p        |    16777 | {52}
 p1_6               | p        |    16777 | {62}
 p1_7               | p        |    16777 | {72}
 p1_8               | p        |    16777 | {82}
 p1_9               | p        |    16777 | {92}
 p1_10              | p        |    16777 | {100}
 p2                 | p        |    16777 | {200}
 p3                 | p        |    16777 | {300}
 p4                 | p        |    16777 | {400}
 p5                 | p        |    16777 | {500}
 p6                 | p        |    16777 | {600}
(17 rows)
```

2.3.3 方法三：INTERVAL

语法：从句指定了 INTERVAL 子句的语法格式，范围分区策略的分区键仅支持 1 列。

NTERVAL ('interval_expr') [STORE IN (tablespace_name [, ...])]

· 列存表不支持间隔分区

· interval_expr：自动创建分区的间隔，例如：1 day、1 month。

· STORE IN (tablespace_name [, ...])：指定存放自动创建分区的表空间列表，如果有指定，则自动创建的分区从表空间列表中循环选择使用，否则使用分区表默认的表空间。

实例：

```
create table partition_orders_3(
o_orderkey integer,
o_custkey integer,
o_orderstatus char(1),
o_totalprice decimal(15,2),
o_orderdate date,
o_orderpriority char(15),
o_clerk char(15),
o_shippriority integer,
o_comment varchar(79),
PRIMARY KEY (o_orderkey)
```

```
)
partition by range(o_orderdate)
interval('1 day')
(
partition p1 values less than('2021-03-08 00:00:00'),
partition p2 values less than('2021-03-09 00:00:00')
);
```

此分区表，一开始创建的分区只有 p1,p2。但如果向表中插入键值不在已有分区范围内的元组，比如：(1，'2021-03-11 00:00:00')，则会自动创建一个分区，其上界与插入的元组间隔 1 day，该元组存入该分区。

```
[omm@db1 openGauss]$ gsql -d tpch -p 26000 -f task2.sql
CREATE TABLE
total time: 3  ms
```

通过 select 命令列出各分区情况

```
tpch=# select relname,parttype,parentid,boundaries from pg_partition where parentid in(select oid from pg_class where relname='partition_orders_3');
      relname       | parttype | parentid |  boundaries
--------------------+----------+----------+--------------
 partition_orders_3 | r        |    16815 |
 p1                 | p        |    16815 | {2021-03-08}
 p2                 | p        |    16815 | {2021-03-09}
(3 rows)
```

插入元组（1，'2021-03-11 00:00:00'）

```
tpch=# insert into partition_orders_3 values（1，'2021-03-11 00:00:00'）;
```

再次通过 select 命令列出各分区情况

```
tpch=# select relname,parttype,parentid,boundaries from pg_partition where parentid in(select oid from pg_class
where relname=' partition_orders_3' );
```

```
tpch=# select relname,parttype,parentid,boundaries from pg_partition where paren
tid in(select oid from pg_class where relname='partition_orders_3');
      relname       | parttype | parentid |       boundaries
--------------------+----------+----------+-------------------------
 partition_orders_3 | r        |    16815 |
 p1                 | p        |    16815 | {2021-03-08}
 p2                 | p        |    16815 | {2021-03-09}
 sys_p1             | p        |    16815 | {"2021-03-12 00:00:00"}
(4 rows)
```

可见，自动生成了 1 个新的分区，上界为 '2021-03-12 00:00:00'，与插入元组间隔为 1 day。

2.3.4 设置分区所在的表空间

在创建分区表时，可以把分区表的不同分区设置在不同的表空间，从而提升整个系统的性能
通过在 partition 语句后面加上 tablespace 来指定创建的分区所在的表空间
实例：

```
create table partition_orders_4(
o_orderkey integer,
o_custkey integer,
o_orderstatus char(1),
o_totalprice decimal(15,2),
o_orderdate date,
o_orderpriority char(15),
```

```
o_clerk char(15),
o_shippriority integer,
o_comment varchar(79),
PRIMARY KEY (o_orderkey)

)
tablespace example2
partition by range(o_orderkey)
(
partition p1 values less than(100),
partition p2 values less than(200) tablespace example4,
partition p3 values less than(300),
partition p4 values less than(maxvalue)
);
```

创建成功

```
[omm@db1 openGauss]$ gsql -d tpch -p 26000 -f task2.sql
gsql:task2.sql:21: NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit inde
x "partition_orders_4_pkey" for table "partition_orders_4"
CREATE TABLE
total time: 25  ms
```

此分区表，分区 p1，p3，p4 都在表空间 example2 中，而分区 p2 在表空间 example4 中

2.4 管理分区表

2.4.1 删除，添加，重命名分区

删除分区 p4

```
tpch=# ALTER TABLE partition_orders_4 DROP PARTITION p4;
```

添加分区 p_4

```
tpch=# ALTER TABLE partition_orders_4 ADD PARTITION p_4 VALUES LESS THAN(MAXVALUE);
```

重命名分区 p3

```
tpch=# ALTER TABLE partition_orders_4 RENAME PARTITION p3 TO p_3;
```

修改后的分区，显示如下：

```
tpch=# ALTER TABLE partition_orders_4 DROP PARTITION p4;
ALTER TABLE
tpch=# ALTER TABLE partition_orders_4 ADD PARTITION p_4 VALUES LESS THAN(MAXVALU
E);
ALTER TABLE
tpch=# ALTER TABLE partition_orders_4 RENAME PARTITION p3 TO p_3;
ALTER TABLE
tpch=# select relname,parttype,parentid,boundaries from pg_partition where paren
tid in(select oid from pg_class where relname='partition_orders_4');
      relname       | parttype | parentid | boundaries
--------------------+----------+----------+------------
 partition_orders_4 | r        |    16822 |
 p1                 | p        |    16822 | {100}
 p2                 | p        |    16822 | {200}
 p_4                | p        |    16822 | {NULL}
 p_3                | p        |    16822 | {300}
(5 rows)
```

2.4.2 修改分区的表空间

将分区 p1 由原来所在的表空间 example2 移动到 example4

```
tpch=# ALTER TABLE partition_orders_4 MOVE PARTITION p1 TABLESPACE example4;
```

```
tpch=# alter table partition_table_4 move partition p1 tablespace example4;
ALTER TABLE
tpch=# select relname,parttype,parentid,boundaries from pg_partition where paren
tid in(select oid from pg_class where relname='partition_table_4');
      relname       | parttype | parentid | boundaries
--------------------+----------+----------+------------
 partition_table_4 | r        |    16665 |
 p2                | p        |    16665 | {200}
 p_4               | p        |    16665 | {NULL}
 p_3               | p        |    16665 | {300}
 p1                | p        |    16665 | {100}
(5 rows)
```

2.4.3 查询分区

查询分区表的分区情况

```
tpch=# select relname,parttype,parentid,boundaries from pg_partition where parentid in(select oid from pg_class
where relname='分区表名称');
```

查询单独分区内的数据，比如：分区 p2

```
tpch=# SELECT * FROM partition_orders_4 PARTITION(p2);
```

或者

```
tpch=# SELECT * FROM partition_orders_4 PARTITION FOR(150);
```

```
tpch=# SELECT * FROM partition_table_4 PARTITION(p2);
 id | col
----+-----
(0 rows)

tpch=# SELECT * FROM partition_table_4 PARTITION FOR(150);
 id | col
----+-----
(0 rows)
```

说明：选择分区有两种方法，一是 partition(分区名称)，二是 partition for（数值），此括号内的数值为所选分区范围内的任意值，如果定义分区时分区键不只一个，那么此括号内的数值个数应该与定义分区时使用的分区键个数相同，并且一一对应。

2.4.4 数据转移

进行交换的普通表和分区必须满足如下条件：

· 普通表和分区的列数目相同，对应列的信息严格一致，包括：列名、列的数据类型、列约束、列的 Collation 信息、列的存储参数、列的压缩信息等。

· 普通表和分区的表压缩信息严格一致。

· 普通表和分区的分布列信息严格一致。

· 普通表和分区的索引个数相同，且对应索引的信息严格一致。

· 普通表和分区的表约束个数相同，且对应表约束的信息严格一致。

· 普通表不可以是临时表。

实例：

查看 orders 表中的数据量

```
select count（*） from orders;
```

```
tpch=# select count(*) from orders;
 count
--------
 300000
(1 row)
```

将 orders 表中的数据转移到分区表 partition_orders_4 中

```
insert into partition_orders_4 select * from orders;
```

```
tpch=# insert into partition_orders_4 select * from orders;
INSERT 0 300000
```

查看整个分区表情况，以及各分区情况

```
select count（*） from partition_orders_4;
```

```
tpch=# select count(*) from partition_orders_4;
 count
--------
 300000
(1 row)
```

```
select * from partition_orders_4;
```

```
tpch=# select * from partition_orders_4;
 o_orderkey | o_custkey | o_orderstatus | o_totalprice |     o_orderdate     | o
_orderpriority |     o_clerk     | o_shippriority |      o_comment
------------+-----------+---------------+--------------+---------------------+--
---------------+-----------------+----------------+---------------------
          1 |      7381 | O             |    181585.13 | 2019-01-02 00:00:00 | 5
-LOW           | Clerk#000000951 |              0 | furiously special f
          2 |     15601 | O             |     40736.64 | 2019-12-02 00:00:00 | 1
-URGENT        | Clerk#000000880 |              0 | furiously special f
          3 |     24664 | F             |    221256.05 | 2016-10-14 00:00:00 | 5
-LOW           | Clerk#000000955 |              0 | furiously special f
          4 |     27356 | O             |      5053.65 | 2018-10-11 00:00:00 | 5
-LOW           | Clerk#000000124 |              0 | furiously special f
          5 |      8897 | F             |    198603.73 | 2017-07-30 00:00:00 | 5
-LOW           | Clerk#000000925 |              0 | furiously special f
          6 |     11125 | F             |      3475.35 | 2015-02-21 00:00:00 | 4
-NOT SPECIFIED | Clerk#000000058 |              0 | furiously special f
          7 |      7828 | O             |    201050.68 | 2019-01-10 00:00:00 | 2
-HIGH          | Clerk#000000470 |              0 | furiously special f
         32 |     26012 | O             |    222247.61 | 2018-07-16 00:00:00 | 2
-HIGH          | Clerk#000000616 |              0 | furiously special f
         33 |     13393 | F             |     83484.38 | 2016-10-27 00:00:00 | 3
-MEDIUM        | Clerk#000000409 |              0 | furiously special f
         34 |     12202 | O             |    163243.09 | 2021-07-21 00:00:00 | 3
-MEDIUM        | Clerk#000000223 |              0 | furiously special f
         35 |     25519 | O             |    174658.48 | 2018-10-23 00:00:00 | 4
-NOT SPECIFIED | Clerk#000000259 |              0 | furiously special f
         36 |     23051 | O             |     83979.37 | 2018-11-03 00:00:00 | 1
-URGENT        | Clerk#000000358 |              0 | furiously special f
         37 |     17224 | F             |    149261.54 | 2015-06-04 00:00:00 | 3
-MEDIUM        | Clerk#000000456 |              0 | furiously special f
         38 |     24967 | O             |      6753.73 | 2019-08-22 00:00:00 | 4
-NOT SPECIFIED | Clerk#000000604 |              0 | furiously special f
         39 |     16354 | O             |    150879.93 | 2019-09-21 00:00:00 | 3
-MEDIUM        | Clerk#000000659 |              0 | furiously special f
         64 |      6424 | F             |     41098.12 | 2017-07-16 00:00:00 | 3
```

```
select count（*） from partition_orders_4 partition(p2);
```

```
tpch=# select count(*) from partition_orders_4 partition(p2);
 count
-------
    28
(1 row)
```

```
select * from partition_orders_4 partition(p2);
```

```
tpch=# select * from partition_orders_4 partition(p2);
 o_orderkey | o_custkey | o_orderstatus | o_totalprice |     o_orderdate     | o
_orderpriority |     o_clerk     | o_shippriority |      o_comment
------------+-----------+---------------+--------------+---------------------+--
---------------+-----------------+----------------+---------------------
        100 |     29401 | O             |    198797.26 | 2021-02-28 00:00:00 | 4
-NOT SPECIFIED | Clerk#000000577 |              0 | furiously special f
        101 |      5600 | O             |    176464.57 | 2019-03-18 00:00:00 | 3
-MEDIUM        | Clerk#000000419 |              0 | furiously special f
        102 |       145 | O             |     79442.23 | 2020-05-09 00:00:00 | 2
-HIGH          | Clerk#000000596 |              0 | furiously special f
        103 |      5821 | O             |    134647.84 | 2019-06-21 00:00:00 | 4
-NOT SPECIFIED | Clerk#000000090 |              0 | furiously special f
        128 |     14792 | F             |     67724.39 | 2015-06-16 00:00:00 | 1
-URGENT        | Clerk#000000385 |              0 | furiously special f
        129 |     14227 | F             |    151238.00 | 2015-11-20 00:00:00 | 5
-LOW           | Clerk#000000859 |              0 | furiously special f
        130 |      7393 | F             |    176613.75 | 2015-05-09 00:00:00 | 2
-HIGH          | Clerk#000000036 |              0 | furiously special f
        131 |     18550 | F             |    168451.68 | 2017-06-08 00:00:00 | 3
-MEDIUM        | Clerk#000000625 |              0 | furiously special f
        132 |      5279 | F             |    207341.91 | 2016-06-11 00:00:00 | 3
-MEDIUM        | Clerk#000000488 |              0 | furiously special f
        133 |      8800 | O             |    141313.30 | 2020-11-29 00:00:00 | 1
-URGENT        | Clerk#000000738 |              0 | furiously special f
        134 |      1240 | F             |     85072.11 | 2015-05-02 00:00:00 | 4
-NOT SPECIFIED | Clerk#000000711 |              0 | furiously special f
        135 |     12097 | O             |    201635.24 | 2018-10-21 00:00:00 | 4
-NOT SPECIFIED | Clerk#000000804 |              0 | furiously special f
```

3 索引

3.1 实验介绍

关于本实验

本实验主要描述 openGauss 数据库如何创建和管理索引

实验目的

1.学会在普通表上创建管理索引

2.学会在分区表上创建管理索引

3.2 索引应用背景

索引可以提高数据的访问速度，但同时也增加了插入、更新和删除操作的处理时间。所以是否要为表增加索引，索引建立在哪些字段上，是创建索引前必须要考虑的问题。需要分析应用程序的业务处理、数据使用、经常被用作查询的条件或者被要求排序的字段来确定是否建立索引。

索引经常建立在数据库表中的以下列上：

· 在经常需要搜索查询的列上创建索引，可以加快搜索的速度。
· 在作为主键的列上创建索引，强制该列的唯一性和组织表中数据的排列结构。
· 在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的。
· 在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间。
· 在经常使用 WHERE 子句的列上创建索引，加快条件的判断速度。
· 为经常出现在关键字 ORDER BY、GROUP BY、DISTINCT 后面的字段建立索引。

索引方式：

1.唯一索引：可用于约束索引属性值的唯一性，或者属性组合值的唯一性。如果一个表声明了唯一约束或者主键，则 openGauss 自动在组成主键或唯一约束的字段上创建唯一索引（可能是多字段索引），以实现这些约束。目前，openGauss 只有 B-Tree 可以创建唯一索引。
2.多字段索引：一个索引可以定义在表中的多个属性上。目前，openGauss 中的 B-Tree 支持多字段索引，且最多可在 32 个字段上创建索引（全局分区索引最多支持 31 个字段）。
3.部分索引：建立在一个表的子集上的索引，这种索引方式只包含满足条件表达式的元组。
4.表达式索引：索引建立在一个函数或者从表中一个或多个属性计算出来的表达式上。表达式索引只有在查询时使用与创建时相同的表达式才会起作用。

注意：

- 索引创建成功后，系统会自动判断何时引用索引。当系统认为使用索引比顺序扫描更快时，就会使用索引。
- 索引创建成功后，必须和表保持同步以保证能够准确地找到新数据，这样就增加了数据操作的负荷。因此请定期删除无用的索引。

3.3 普通表上创建管理索引

3.3.1 创建索引

步骤 1：创建 orders 表的备份 orderscopy2，并将数据导入

```
create table orderscopy2(
o_orderkey integer,
o_custkey integer,
o_orderstatus char(1),
o_totalprice decimal(15,2),
o_orderdate date,
o_orderpriority char(15),
o_clerk char(15),
o_shippriority integer,
```

```
o_comment varchar(79),
PRIMARY KEY (o_orderkey)
)
tablespace example2;

INSERT INTO orderscopy2
SELECT *
FROM orders;
```

步骤 2：创建普通索引

如果对于 orderscopy2 表，需要经常进行以下查询。

```
tpch=# SELECT * FROM orderscopy2 WHERE o_totalprice>10000;
```

使用以下命令创建索引。

```
tpch=# CREATE INDEX orderscopy2_index_totalprice ON orderscopy2(o_totalprice);
```

步骤 3：创建多字段索引

假如用户需要经常查询表 orderscopy2 中 o_orderstatus 是’0’，且 o_totalprice 小于 10000 的记录，使用以下命令进行查询。

```
tpch=# SELECT * FROM orderscopy2 WHERE o_orderstatus=’0’ AND o_totalprice<10000;
```

使用以下命令在字段 o_orderstatus 和 o_totalprice 上定义一个多字段索引。

```
tpch=# CREATE INDEX orderscopy2_index_more_column ON orderscopy2(o_orderstatus, o_totalprice);
```

步骤 4：创建部分索引

如果只需要查询 o_orderstatus 为’0’的记录

```
tpch=# SELECT * FROM orderscopy2 WHERE o_orderstatus=’0’;
```

可以创建部分索引来提升查询效率。

```
tpch=# CREATE INDEX orderscopy2_part_index ON orderscopy2(o_orderstatus) WHERE o_orderstatus=’0’;
```

步骤 5：创建表达式索引

假如经常需要查询代号以 8 结尾的收银员的信息，执行如下命令进行查询。

```
tpch=# SELECT * FROM orderscopy2 WHERE o_clerk like ’%8’;
```

可以为上面的查询创建表达式索引：

```
tpch=# CREATE INDEX orderscopy2_para_index ON orderscopy2(reverse(o_clerk) varchar_pattern_ops);
```

3.3.2 管理索引

步骤 1：查询索引

执行如下命令查询系统和用户定义的所有索引。

```
tpch=# SELECT RELNAME FROM PG_CLASS WHERE RELKIND='i';
```

可以看到上面创建的索引

```
 orderscopy2_index_totalprice
 orderscopy2_index_more_column
 orderscopy2_part_index
 lineitemcopy2_pk
 lineitemcopy2_uk
 lineitem_index1
 orderscopy2_para_index
```

\di+可以查询用户定义的所有索引

```
tpch=# \di+
                                                    List of relations
 Schema |             Name              | Type  | Owner |       Table        |
  Size    | Storage | Description
--------+-------------------------------+-------+-------+--------------------+--
---------+---------+-------------
 public | customer_index                | index | omm   | customer_new       | 1
480 kB   |         |
 public | customer_pkey                 | index | omm   | customer           | 6
80 kB    |         |
 public | customercopy1_pkey            | index | omm   | customercopy1      | 6
72 kB    |         |
 public | lineitem_index1               | index | omm   | lineitem_new       | 2
6 MB     |         |
 public | lineitem_index2               | index | omm   | lineitem_new       | 2
6 MB     |         |
 public | lineitem_pkey                 | index | omm   | lineitem           | 2
6 MB     |         |
 public | lineitemcopy1_pkey            | index | omm   | lineitemcopy1      | 7
7 MB     |         |
 public | lineitemcopy2_pk              | index | omm   | lineitemcopy2      | 8
192 bytes |         |
 public | lineitemcopy2_uk              | index | omm   | lineitemcopy2      | 8
192 bytes |         |
 public | nation_pkey                   | index | omm   | nation             | 1
6 kB     |         |
 public | orders_pkey                   | index | omm   | orders             | 6
616 kB   |         |
 public | orderscopy1_pkey              | index | omm   | orderscopy1        | 6
600 kB   |         |
 public | orderscopy2_index_more_column | index | omm   | orderscopy2        | 9
264 kB   |         |
 public | orderscopy2_index_totalprice  | index | omm   | orderscopy2        | 9
192 kB   |         |
 public | orderscopy2_para_index        | index | omm   | orderscopy2        | 9
264 kB   |         |
 public | orderscopy2_part_index        | index | omm   | orderscopy2        | 3
240 kB   |         |
```

执行如下命令查询指定索引的信息。

```
tpch=# \di+ orderscopy2_index_totalprice
```

```
tpch=# \di+ orderscopy2_index_totalprice;
                                        List of relations
 Schema |             Name             | Type  | Owner |    Table    |  Size   |
 Storage | Description
--------+------------------------------+-------+-------+-------------+---------+
---------+-------------
 public | orderscopy2_index_totalprice | index | omm   | orderscopy2 | 9192 kB |
        |
(1 row)
```

步骤 2：重命名索引

执行如下命令对索引 orderscopy2_index_more_column 重命名 orderscopy2_index_orderstatus_totalprice。

```
tpch=# ALTER INDEX orderscopy2_index_more_column RENAME TO orderscopy2_index_orderstatus_totalprice;
```

用\di+查看，索引重命名成功

```
 public | orderscopy2_index_orderstatus_totalprice | index | omm   | orderscopy2
        | 9264 kB    |         |
 public | orderscopy2_index_totalprice             | index | omm   | orderscopy2
        | 9192 kB    |         |
 public | orderscopy2_para_index                   | index | omm   | orderscopy2
        | 9264 kB    |         |
 public | orderscopy2_part_index                   | index | omm   | orderscopy2
        | 3240 kB    |         |
```

步骤 3：删除索引

删除 orderscopy2_para_index 索引

```
tpch=# DROP INDEX orderscopy2_para_index;
```

删除成功

```
 public | orderscopy2_index_orderstatus_totalprice | index | omm   | orderscopy2
        | 9264 kB    |         |
 public | orderscopy2_index_totalprice             | index | omm   | orderscopy2
        | 9192 kB    |         |
 public | orderscopy2_part_index                   | index | omm   | orderscopy2
        | 3240 kB    |         |
```

3.4 分区表上创建管理索引

3.4.1 创建索引

分区表索引分为 LOCAL 索引与 GLOBAL 索引，一个 LOCAL 索引对应一个具体分区(有多少个分区就有多少个索引文件)，而 GLOBAL 索引则对应整个分区表（只有一个索引文件）。

步骤 1：以 orders 表为例创建一个分区表

```
create table partition_orders_0(
o_orderkey integer,
o_custkey integer,
o_orderstatus char(1),
o_totalprice decimal(15,2),
```

```
o_orderdate date,
o_orderpriority char(15),
o_clerk char(15),
o_shippriority integer,
o_comment varchar(79),
PRIMARY KEY (o_orderkey)
)
tablespace example2
partition by range(o_orderkey)
(
partition p1 values less than(100),
partition p2 values less than(200),
partition p3 values less than(300),
partition p4 values less than(maxvalue)
);
```

步骤 2：创建 GLOBAL 索引

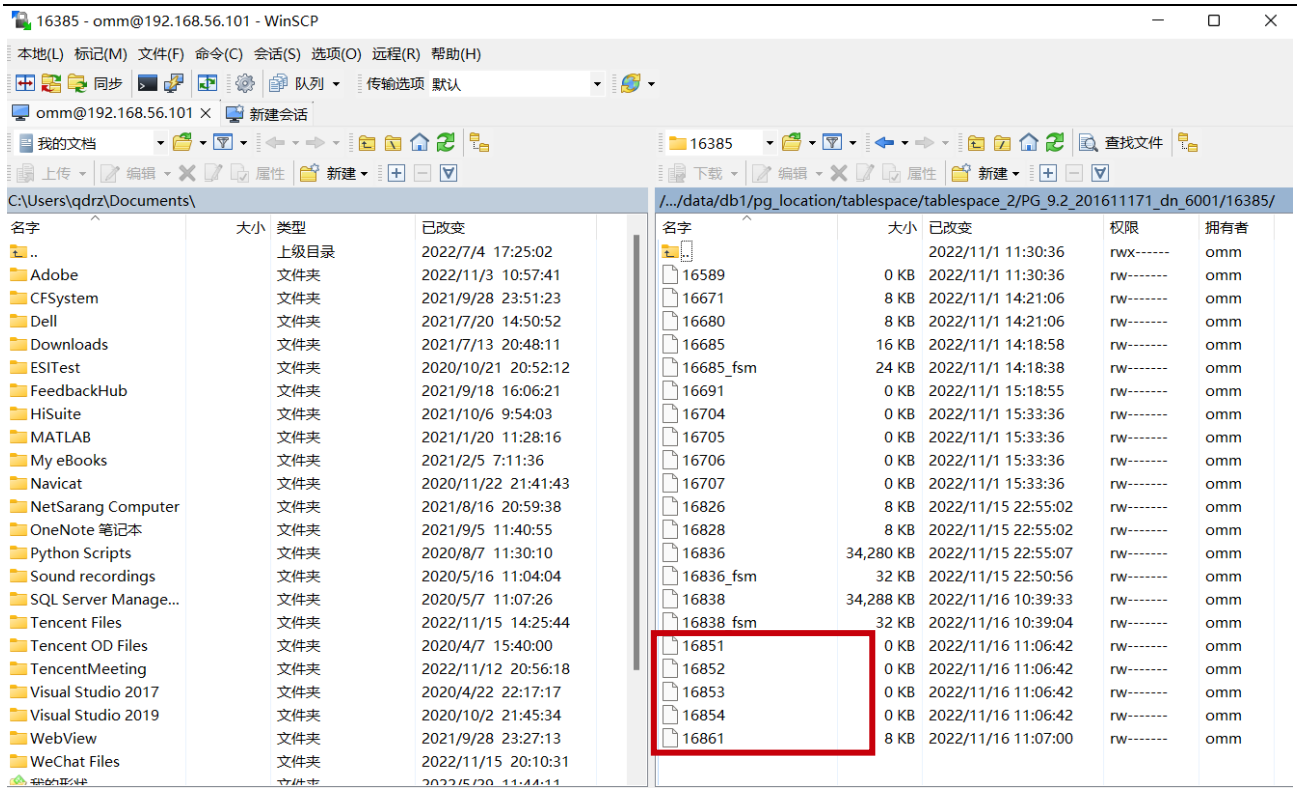
在 o_custkey 上创建分区表 GLOBAL 索引，存储在表空间 example2 上

```
create index global_index_custkey on partition_orders_0(o_custkey) global tablespace example2;
```

创建成功后

```
tpch=# \di+
                                                    List of relations
 Schema |                 Name                 |          Type          | Owner |       Table        |    Size    | Storage | Description
--------+--------------------------------------+------------------------+-------+--------------------+------------+---------+-------------
 public | customer_index                       | index                  | omm   | customer_new       | 1480 kB    |         |
 public | customer_pkey                        | index                  | omm   | customer           | 680 kB     |         |
 public | customercopy1_pkey                   | index                  | omm   | customercopy1      | 672 kB     |         |
 public | global_index_custkey                 | global partition index | omm   | partition_orders_0 | 8192 bytes |         |
 public | lineitem_index1                      | index                  | omm   | lineitem_new       | 26 MB      |         |
 public | lineitem_index2                      | index                  | omm   | lineitem_new       | 26 MB      |         |
 public | lineitem_pkey                        | index                  | omm   | lineitem           | 26 MB      |         |
 public | lineitemcopy1_pkey                   | index                  | omm   | lineitemcopy1      | 77 MB      |         |
 public | lineitemcopy2_pk                     | index                  | omm   | lineitemcopy2      | 8192 bytes |         |
 public | lineitemcopy2_uk                     | index                  | omm   | lineitemcopy2      | 8192 bytes |         |
 public | local_index_shippriority             | index                  | omm   | partition_orders_0 | 32 kB      |         |
 public | local_index_totalprice               | index                  | omm   | partition_orders_0 | 32 kB      |         |
 public | nation_pkey                          | index                  | omm   | nation             | 16 kB      |         |
 public | orders_pkey                          | index                  | omm   | orders             | 6616 kB    |         |
```

用 WinSCP 可以看到，只有一个索引文件（前面 4 个是上面创建的 partition_orders_0 分区表，4 个分区对应 4 个文件）



步骤 3：创建不指定索引分区名称的 LOCAL 索引

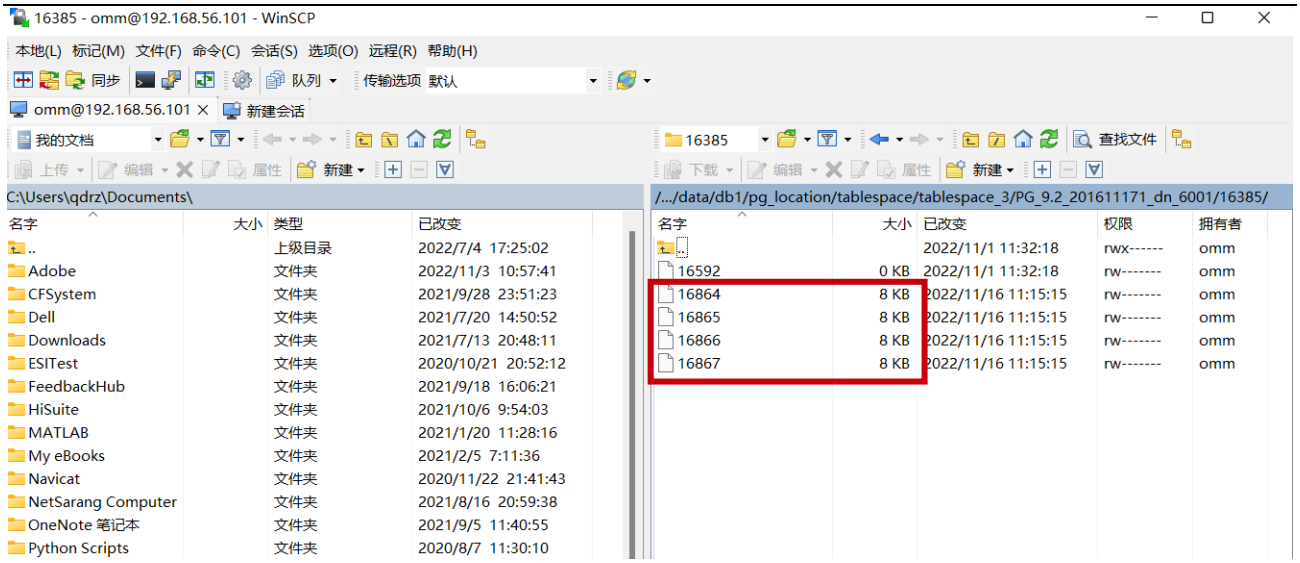
在 o_totalprice 上创建分区表 LOCAL 索引，不指定索引分区的名称，存储在 example3 上（原表与索引可以分别存储在不同的表空间上）

```
create index local_index_totalprice on partition_orders_0(o_totalprice) local tablespace example3;
```

创建成功后

```
tpch=# \di+
                                                    List of relations
 Schema |                  Name                  |          Type          | Owner |      Table       |    Size    | Storage | Description
--------+----------------------------------------+------------------------+-------+------------------+------------+---------+-------------
 public | customer_index                         | index                  | omm   | customer_new     | 1480 kB    |         |
 public | customer_pkey                          | index                  | omm   | customer         | 680 kB     |         |
 public | customercopy1_pkey                     | index                  | omm   | customercopy1    | 672 kB     |         |
 public | global_index_custkey                   | global partition index | omm   | partition_orders_0 | 8192 bytes |         |
 public | lineitem_index1                        | index                  | omm   | lineitem_new     | 26 MB      |         |
 public | lineitem_index2                        | index                  | omm   | lineitem_new     | 26 MB      |         |
 public | lineitem_pkey                          | index                  | omm   | lineitem         | 26 MB      |         |
 public | lineitemcopy1_pkey                     | index                  | omm   | lineitemcopy1    | 77 MB      |         |
 public | lineitemcopy2_pk                       | index                  | omm   | lineitemcopy2    | 8192 bytes |         |
 public | lineitemcopy2_uk                       | index                  | omm   | lineitemcopy2    | 8192 bytes |         |
 public | local_index_totalprice                 | index                  | omm   | partition_orders_0 | 32 kB      |         |
 public | nation_pkey                            | index                  | omm   | nation           | 16 kB      |         |
 public | orders_pkey                            | index                  | omm   | orders           | 6616 kB    |         |
 public | orderscopy1_pkey                       | index                  | omm   | orderscopy1      | 6600 kB    |         |
 public | orderscopy2_index_orderstatus_totalprice | index                  | omm   | orderscopy2      | 9264 kB    |         |
 public | orderscopy2_index_totalprice           | index                  | omm   | orderscopy2      | 9192 kB    |         |
 public | orderscopy2_part_index                 | index                  | omm   | orderscopy2      | 3240 kB    |         |
 public | orderscopy2_pkey                       | index                  | omm   | orderscopy2      | 6600 kB    |         |
```

用 WinSCP 看出，有 4 个索引文件（因为有 4 个分区）



步骤 4：创建指定索引分区名称的 LOCAL 索引

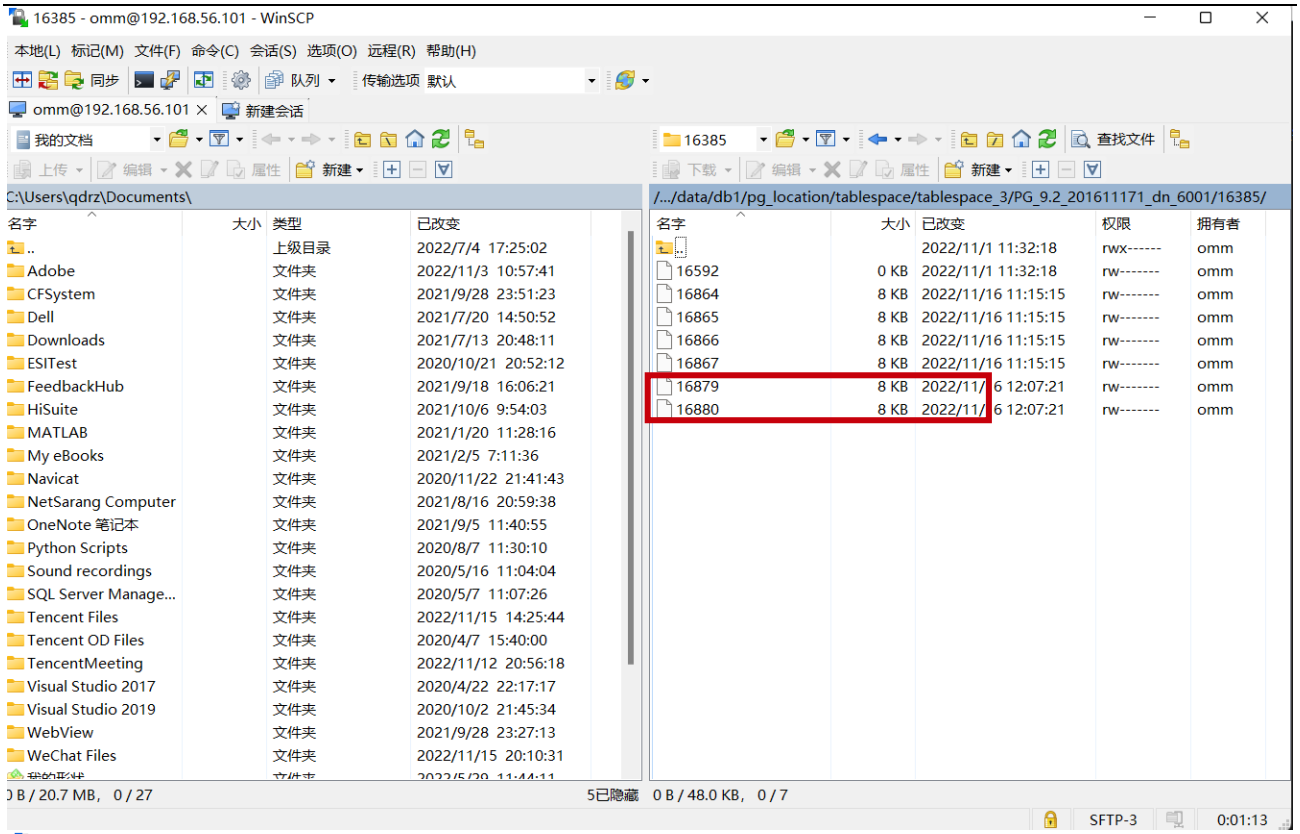
在 o_shippriority 上创建分区表 LOCAL 索引，指定索引分区的名称，p1,p2 分区索引存储在 example3 上，p3,p4f 分区索引存储在 example4 上（不同分区的索引可以分别存储在不同的表空间上）

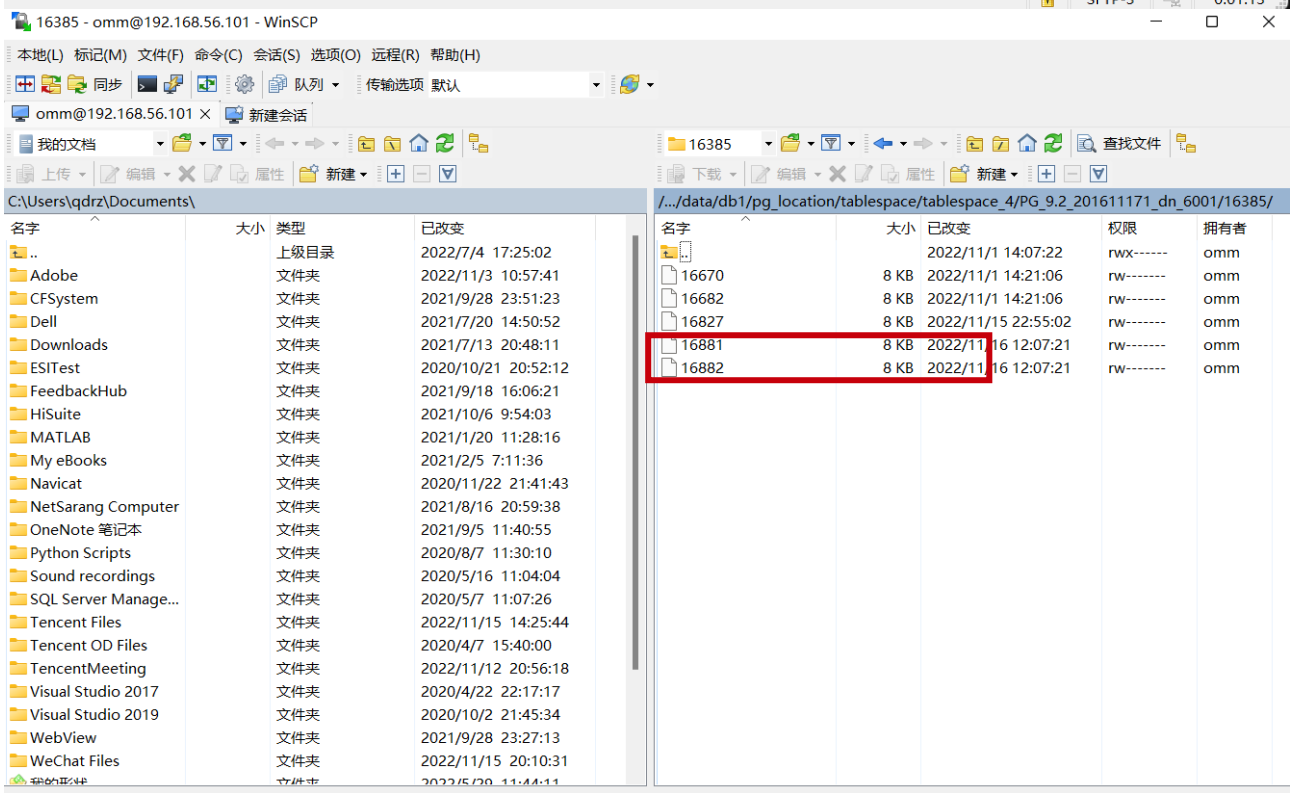
```
create index local_index_shippriority on partition_orders_0(o_shippriority) local
(
 partition p1_index,
 partition p2_index,
 partition p3_index tablespace example4,
 partition p4_index tablespace example4
) tablespace example3;
```

创建成功后

```
tpch=# \di+
                                                    List of relations
 Schema |                    Name                    |         Type          | Owner |      Table       |    Size    | Storage | Description
--------+--------------------------------------------+-----------------------+-------+------------------+------------+---------+-------------
 public | customer_index                             | index                 | omm   | customer_new     | 1480 kB    |         |
 public | customer_pkey                              | index                 | omm   | customer         | 680 kB     |         |
 public | customercopy1_pkey                         | index                 | omm   | customercopy1    | 672 kB     |         |
 public | global_index_custkey                       | global partition index | omm   | partition_orders_0 | 8192 bytes |         |
 public | lineitem_index1                            | index                 | omm   | lineitem_new     | 26 MB      |         |
 public | lineitem_index2                            | index                 | omm   | lineitem_new     | 26 MB      |         |
 public | lineitem_pkey                              | index                 | omm   | lineitem         | 26 MB      |         |
 public | lineitemcopy1_pkey                         | index                 | omm   | lineitemcopy1    | 77 MB      |         |
 public | lineitemcopy2_pk                           | index                 | omm   | lineitemcopy2    | 8192 bytes |         |
 public | lineitemcopy2_uk                           | index                 | omm   | lineitemcopy2    | 8192 bytes |         |
 public | local_index_shippriority                   | index                 | omm   | partition_orders_0 | 32 kB      |         |
 public | local_index_totalprice                     | index                 | omm   | partition_orders_0 | 32 kB      |         |
 public | nation_pkey                                | index                 | omm   | nation           | 16 kB      |         |
 public | orders_pkey                                | index                 | omm   | orders           | 6616 kB    |         |
 public | orderscopy1_pkey                           | index                 | omm   | orderscopy1      | 6600 kB    |         |
 public | orderscopy2_index_orderstatus_totalprice   | index                 | omm   | orderscopy2      | 9264 kB    |         |
 public | orderscopy2_index_totalprice               | index                 | omm   | orderscopy2      | 9192 kB    |         |
 public | orderscopy2_part_index                     | index                 | omm   | orderscopy2      | 3240 kB    |         |
 public | orderscopy2_pkey                           | index                 | omm   | orderscopy2      | 6600 kB    |         |
```

用 WinSCP 看到，分别在 example3 有 2 个索引文件和 example4 有 2 个索引文件


16385 - omm@192.168.56.101 - WinSCP
本地(L) 标记(M) 文件(F) 命令(C) 会话(S) 选项(O) 远程(R) 帮助(H)
同步 队列 传输选项 默认
omm@192.168.56.101 新建会话
我的文档 16385 查找文件
上传 编辑 属性 新建 下载 编辑 属性 新建
C:\Users\qdrz\Documents\
/.../data/db1/pg_location/tablespace/tablespace_3/PG_9.2_201611171_dn_6001/16385/
名字 大小 类型 已改变
名字 大小 已改变 权限 拥有者
.. 上级目录 2022/7/4 17:25:02
Adobe 文件夹 2022/11/3 10:57:41
CFSystem 文件夹 2021/9/28 23:51:23
Dell 文件夹 2021/7/20 14:50:52
Downloads 文件夹 2021/7/13 20:48:11
ESITest 文件夹 2020/10/21 20:52:12
FeedbackHub 文件夹 2021/9/18 16:06:21
HiSuite 文件夹 2021/10/6 9:54:03
MATLAB 文件夹 2021/1/20 11:28:16
My eBooks 文件夹 2021/2/5 7:11:36
Navicat 文件夹 2020/11/22 21:41:43
NetSarang Computer 文件夹 2021/8/16 20:59:38
OneNote 笔记本 文件夹 2021/9/5 11:40:55
Python Scripts 文件夹 2020/8/7 11:30:10
Sound recordings 文件夹 2020/5/16 11:04:04
SQL Server Manage... 文件夹 2020/5/7 11:07:26
Tencent Files 文件夹 2022/11/15 14:25:44
Tencent OD Files 文件夹 2020/4/7 15:40:00
TencentMeeting 文件夹 2022/11/12 20:56:18
Visual Studio 2017 文件夹 2020/4/22 22:17:17
Visual Studio 2019 文件夹 2020/10/2 21:45:34
WebView 文件夹 2021/9/28 23:27:13
WeChat Files 文件夹 2022/11/15 20:10:31
2022/11/1 11:32:18 rwx------ omm
16592 0 KB 2022/11/1 11:32:18 rw------- omm
16864 8 KB 2022/11/16 11:15:15 rw------- omm
16865 8 KB 2022/11/16 11:15:15 rw------- omm
16866 8 KB 2022/11/16 11:15:15 rw------- omm
16867 8 KB 2022/11/16 11:15:15 rw------- omm
16879 8 KB 2022/11/16 12:07:21 rw------- omm
16880 8 KB 2022/11/16 12:07:21 rw------- omm
B / 20.7 MB, 0 / 27
5已隐藏 0 B / 48.0 KB, 0 / 7
SFTP-3 0:01:13


16385 - omm@192.168.56.101 - WinSCP
本地(L) 标记(M) 文件(F) 命令(C) 会话(S) 选项(O) 远程(R) 帮助(H)
同步 队列 传输选项 默认
omm@192.168.56.101 新建会话
我的文档 16385 查找文件
上传 编辑 属性 新建 下载 编辑 属性 新建
C:\Users\qdrz\Documents\
/.../data/db1/pg_location/tablespace/tablespace_4/PG_9.2_201611171_dn_6001/16385/
名字 大小 类型 已改变
名字 大小 已改变 权限 拥有者
.. 上级目录 2022/7/4 17:25:02
Adobe 文件夹 2022/11/3 10:57:41
CFSystem 文件夹 2021/9/28 23:51:23
Dell 文件夹 2021/7/20 14:50:52
Downloads 文件夹 2021/7/13 20:48:11
ESITest 文件夹 2020/10/21 20:52:12
FeedbackHub 文件夹 2021/9/18 16:06:21
HiSuite 文件夹 2021/10/6 9:54:03
MATLAB 文件夹 2021/1/20 11:28:16
My eBooks 文件夹 2021/2/5 7:11:36
Navicat 文件夹 2020/11/22 21:41:43
NetSarang Computer 文件夹 2021/8/16 20:59:38
OneNote 笔记本 文件夹 2021/9/5 11:40:55
Python Scripts 文件夹 2020/8/7 11:30:10
Sound recordings 文件夹 2020/5/16 11:04:04
SQL Server Manage... 文件夹 2020/5/7 11:07:26
Tencent Files 文件夹 2022/11/15 14:25:44
Tencent OD Files 文件夹 2020/4/7 15:40:00
TencentMeeting 文件夹 2022/11/12 20:56:18
Visual Studio 2017 文件夹 2020/4/22 22:17:17
Visual Studio 2019 文件夹 2020/10/2 21:45:34
WebView 文件夹 2021/9/28 23:27:13
WeChat Files 文件夹 2022/11/15 20:10:31
2022/11/1 14:07:22 rwx------ omm
16670 8 KB 2022/11/1 14:21:06 rw------- omm
16682 8 KB 2022/11/1 14:21:06 rw------- omm
16827 8 KB 2022/11/15 22:55:02 rw------- omm
16881 8 KB 2022/11/16 12:07:21 rw------- omm
16882 8 KB 2022/11/16 12:07:21 rw------- omm

3.4.2 管理索引

步骤 1：修改索引分区所在的表空间

将分区索引 p1_index 从 example3 移到 example2

```
tpch=# ALTER INDEX local_index_shippriority MOVE PARTITION p1_index TABLESPACE example2;
```

```
tpch=# ALTER INDEX local_index_shippriority MOVE PARTITION p1_index TABLESPAC
E example2;
ALTER INDEX
```

步骤 2：重命名索引分区

将分区索引 p2_index 重命名为 p2_index_new

```
tpch=# ALTER INDEX local_index_shippriority RENAME PARTITION p2_index TO p2_index_new;
```

```
tpch=# ALTER INDEX local_index_shippriority RENAME PARTITION p2_index TO p2_i
ndex_new;
ALTER INDEX
```

步骤 3：删除索引

要删除索引只能删除整个索引，不能删除单独的分区索引

```
tpch=# drop index global_index_custkey;
tpch=# drop index local_index_totalprice;
tpch=# drop index local_index_shippriority;
```

4 OpenGauss 段页式特性

4.1 实验介绍

（该部分实验需在云平台或高版本 openGauss 上进行）

openGauss 通用的普通表，每个数据表对应一个逻辑逻辑上的大文件（最大 32T），该逻辑文件又按照固定的大小划分多个实际文件存在对应的数据库目录下面。所以，每张数据表随着数据量的增多，底层的数据存储所需文件数量会逐渐增多。同时，openGauss 对外提供 hashbucket 表、大分区表等特性，每张数据表会被拆分为若干个子表，底层所需文件数量更是成倍增长。由此，这种存储管理模式存在以下问题：

1. 对文件系统依赖大，无法进行细粒度的控制提升可维护性；
2. 大数据量下文件句柄过多，目前只能依赖虚拟句柄来解决，影响系统性能；
3. 小文件数量过多会导致全量 build、全量备份等场景下的随机 IO 问题，影响性能；

为了解决以上问题，openGauss 引入段页式存储管理机制，类似于操作系统的段页式内存管理，但是在实现机制上区别很大。

本实验通过在 TPC-H 测试基准的 orders 表的备份表上应用段页式存储的相关函数操作，观察应用段页式存储后的效果。

4.2 段页式实现原理

借鉴操作系统内存管理中的段页式管理（segmentation with paging）机制，DBMS 将数据库文件在外存上的存储空间在逻辑上组织为段-页方式。

在段页式存储管理下，表空间和数据文件以段(Segment)、区(Extent)以及页（Page/Block）为逻辑组织方式进行存储的分配和管理。如下图所示。具体来说，一个 database(在一个 tablespace 中)有且仅有一个段空间（segment space），实际物理存储可以是一个文件，也可以拆分成多个文件。该 database 中所有 table 都从该段空间中分配数据。所以表的个数和实际物理文件个数无关。每个 table 有一个逻辑上的 segment，该 table 所有的数据都存在该 segment 上。每个 segment 会挂载多个 extent，每个 extent 是一块连续的物理页。Extent 的大小可以根据业务需求灵活调整，避免存储空间的浪费。

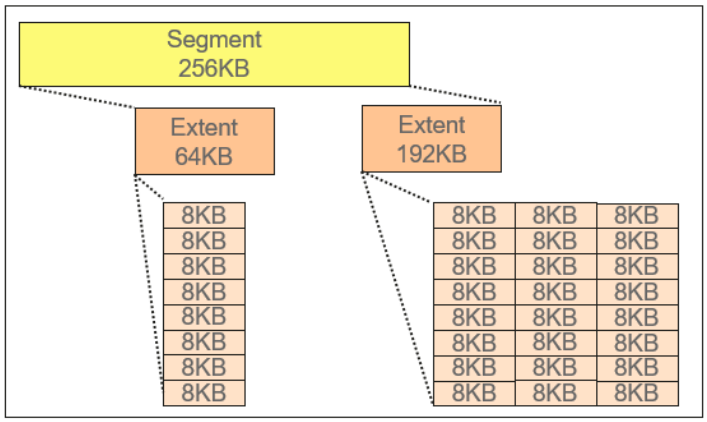


图 1 段页式存储设计示意图

段页式文件可以自动扩容，不需要用户手动指定，直到磁盘空间用满或者达到 tablespace 设置的 limit 限制。段页式存储不会自动回收磁盘空间。当某些数据表被删除之后，其在段页式文件中占据的空间，会被保留，即段页式文件中会存在一些空洞，磁盘空间没有被释放。这些空洞会被后面新扩展或者创建出来的表重用。用户如果确定不需要重用这些空洞，可以手动调用系统函数，来进行磁盘空间的回收，释放磁盘空间。

内部实现上，每个 segment 对应原先页式存储的一个物理文件。比如每个分区表、每个 hashbucket 表的一个 bucket，都会有一个单独的 segment。每个 segment 下面会挂载多个 extent，每个 extent 在文件中是连续的，但 extent 之间未必连续。Segment 会动态扩展，加入新的 extent，但不能直接回收某个 extent。可以通过对整个 table 做 truncate 或者 cluster 等方式，以 segment 为粒度回收存储空间。

目前支持四种大小的 extent，分别是 64K/1M/8M/64M。对于一个 segment 来说，每一次扩展的 extent 的大小是固定的。前 16 个 extent 大小为 64K，第 17 到第 143 个 extent 大小为 1MB，依次类推。具体参数如下表所示。

表 1 segment 存储 extent 分类表

Group	Extent size	Extent page count	Extent count range	Total page count	Total size
1	64K	8	[1,16]	128	1M
2	1M	128	[17,143]	16K	128M
3	8M	1024	[144,255]	128K	1G
4	64M	8192	[256, …]	…	…

4.3 段页式表使用指导

用户在用 SQL 语句 create table 建表时可以通过指定参数 segment=on，使得行存表可以使用段页式的方式存储数据。目前段页式存储不支持列存表。段页式表空间是自动创建的，不需要用户有额外的命令。

1. 以订单表 orders 为例，指定参数 segment=on，创建段页式普通表

```
CREATE TABLE orderscopy(
o_orderkey integer,
o_custkey integer,
```

```
o_orderstatus char(1),
o_totalprice decimal(15,2),
o_orderdate date,
o_orderpriority char(15),
o_clerk char(15),
o_shippriority integer,
o_comment varchar(79),
PRIMARY KEY (o_orderkey)
)
with(segment=on);
```

```
CREATE TABLE orderscopy(
o_orderkey integer,
o_custkey integer,
o_orderstatus char(1),
o_totalprice decimal(15,2),
o_orderdate date,
o_orderpriority char(15),
o_clerk char(15),
o_shippriority integer,
o_comment varchar(79),
PRIMARY KEY (o_orderkey)
)
with(segment=on);
执行成功，耗时：[61ms.]

warning:
CREATE TABLE / PRIMARY KEY will create implicit index "orderscopy_pkey" for table "orderscopy"
```

创建成功后，导入 orders 表数据。

为了让用户更好使用段页式功能，openGauss 提供了两个 built in 的系统函数，显示 extent 的使用情况。用户可以使用这两个视图，决定是否回收和回收哪一部分的数据。

- local_segment_space_info(tablespacename TEXT, databasename TEXT)

描述：输出为该表空间下所有 ExtentGroup 的使用信息。

表 2 local_segment_space_info 视图字段信息

名称	描述
node_name	节点名称。
extent_size	该 ExtentGroup 的 extent 规格，单位是 block 数。
forknum	Fork 号。
total_blocks	物理文件总 extent 数目。
meta_data_blocks	表空间管理的 metadata 占用的 block 数，只包括 space

	header、map page 等，不包括 segment head。
used_data_blocks	存数据占用的 extent 数目。包括 segment head。
utilization	使用的 block 数占总 block 数的百分比。即(used_data_blocks+meta_data_block)/total_blocks。
high_water_mark	高水位线，被分配出去的 extent，最大的物理页号。超过高水位线的 block 都没有被使用，可以被直接回收。

以新建 orderscopy 表所在表空间和数据库为例，查看 local_segment_space_info（）函数的输出结果：

```
1 select * from local_segment_space_info('pg_default', 'postgres');
```

select * from local 输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)

	node_nam	extent_size	forknum	total_block	meta_data	used_data	utilization	high_water
1	dn_6001_6002_6	1	0	16,384	4,157	5	0.25402832	4,162
2	dn_6001_6002_6	8	0	16,384	4,157	264	0.26983643	4,421
3	dn_6001_6002_6	8	1	16,384	4,157	8	0.25421143	4,165
4	dn_6001_6002_6	128	0	16,384	4,157	4,992	0.55841064	9,149

- pg_stat_segment_extent_usage(int4 tablespace oid, int4 database oid, int4 extent_type, int4 forknum)

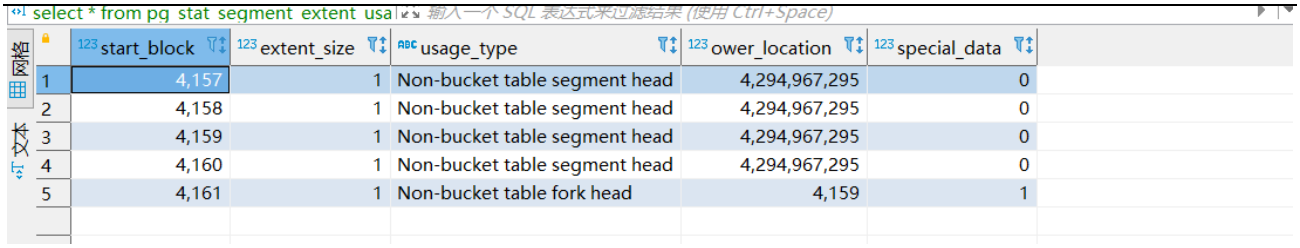
描述：每次返回一个 ExtentGroup 中，每个被分配出去的 extent 的使用情况。extent_type 表示 ExtentGroup 的类型，合理取值为[1,5]的 int 值。在此范围外的会报 error。forknum 表示 fork 号，合法取值为[0,4]的 int 值，目前只有三种值有效，数据文件为 0，FSM 文件为 1，visibility map 文件为 2。

表 3 pg_stat_segment_extent_usage 视图字段信息

名称	描述
start_block	Extent 的起始物理页号。
extent_size	Extent 的大小。
usage_type	Extent 的使用类型，比如 segment head、data extent 等。
ower_location	有指针指向该 extent 的对象的位置。比如 data extent 的 owner 就是它所属的 segment 的 head 位置。
special_data	该 extent 在它 owner 中的位置。该字段的数据跟使用类型有关。比如 data extent 的 special data 就是它在所属 segment 中的 extent id。

查找 orderscopy 表所在表空间和数据库的 oid，查看 pg_stat_segment_extent_usage（）函数的输出结果：

```
select * from pg_stat_segment_extent_usage((select oid::int4 from pg_tablespace where spcname='pg_default'), (select oid::int4 from pg_database where datname='postgres'), 1, 0);
```



	start_block	extent_size	usage_type	ower_location	special_data
1	4,157	1	Non-bucket table segment head	4,294,967,295	0
2	4,158	1	Non-bucket table segment head	4,294,967,295	0
3	4,159	1	Non-bucket table segment head	4,294,967,295	0
4	4,160	1	Non-bucket table segment head	4,294,967,295	0
5	4,161	1	Non-bucket table fork head	4,159	1

```
select * from pg_stat_segment_extent_usage((select oid::int4 from pg_tablespace where spcname='pg_default'), (select oid::int4 from pg_database where datname='postgres'), 2, 0);
```

	start_block	extent_size	usage_type	ower_location	special_data
1	4,157	8	Data extent	4,158	0
2	4,165	8	Data extent	4,160	0
3	4,173	8	Data extent	4,159	0
4	4,181	8	Data extent	4,159	1
5	4,189	8	Data extent	4,159	2
6	4,197	8	Data extent	4,159	3
7	4,205	8	Data extent	4,160	1
8	4,213	8	Data extent	4,159	4
9	4,221	8	Data extent	4,159	5
10	4,229	8	Data extent	4,159	6
11	4,237	8	Data extent	4,159	7
12	4,245	8	Data extent	4,159	8
13	4,253	8	Data extent	4,159	9
14	4,261	8	Data extent	4,160	2
15	4,269	8	Data extent	4,159	10
16	4,277	8	Data extent	4,159	11
17	4,285	8	Data extent	4,159	12
18	4,293	8	Data extent	4,159	13
19	4,301	8	Data extent	4,159	14
20	4,309	8	Data extent	4,160	3
21	4,317	8	Data extent	4,159	15
22	4,325	8	Data extent	4,160	4
23	4,333	8	Data extent	4,160	5

- gs_space_shrink(int4 tablespace, int4 database, int4 extent_type, int4 forknum)

描述：当前节点上对指定段页式空间做物理空间收缩。注意，目前只支持对当前连接的 database 做 shrink。（对指定段页式空间做物理空间收缩）,传入的参数是 tablespace 和 database 的 oid，extent_type 为[2,5]的 int 值。注意：extent_type = 1 表示段页式元数据，目前不支持对元数据所在的物理文件做收缩。该函数仅限工具使用，不建议用户直接使用。