# 《数据库物理设计》

# 实验报告


**学院：** 计算机学院（国家示范性软件学院）

**班级：** 2019211308 2019211308 2019211308

**姓名：** 顾天阳 曾世茂 庞仕泽

**学号：** 2019211539 2019211532 2019211509

目录

# 一、 表空间

# 二、 分区表

## 2.3 创建分区表

(1) 方法一 values less than

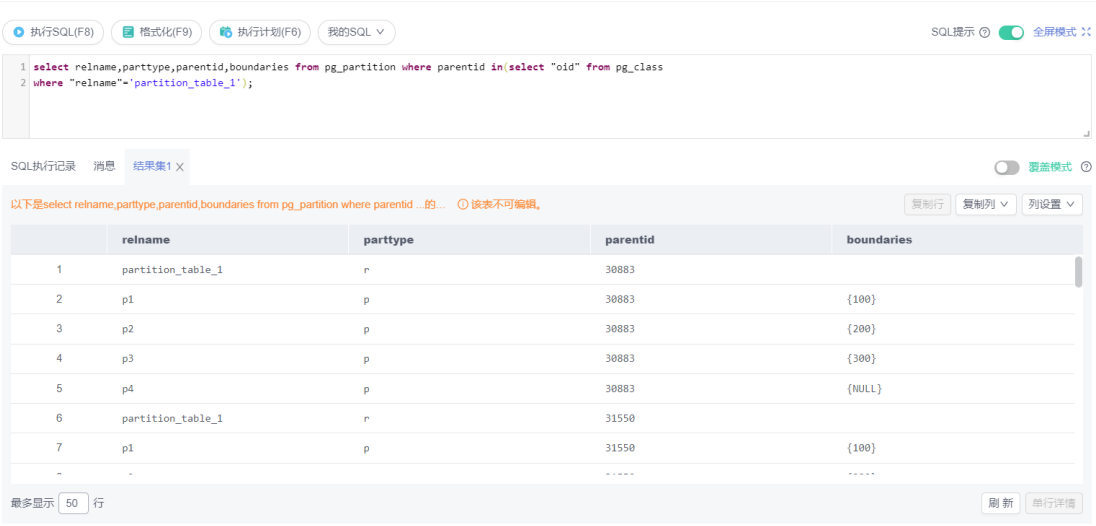
```
CREATE  teble partition_table_1(
    id SERIAL PRIMARY KEY ,
    col VARCHAR (8))
PARTITION  by RANGE (id)(
partition p1 VALUES less than(100),
  PARTITION  p2 VALUES  less than (200),
  PARTITION  p3 VALUES  less than (300),
  PARTITION  p4 VALUES less than(MAXVALUE )
);
```

```
--------------开始执行--------------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
CREATE  table partition_table_1(
        id SERIAL PRIMARY KEY ,
    col VARCHAR (8))
PARTITION  by RANGE (id)(
partition p1 VALUES less than(100),
  PARTITION  p2 VALUES  less than (200),
  PARTITION  p3 VALUES  less than (300),
  PARTITION  p4 VALUES less than(MAXVALUE )
);
执行成功，耗时：[33ms.]

warning:
CREATE TABLE will create implicit sequence "partition_table_1_id_seq" for serial column "partition_table_1.id"
CREATE TABLE / PRIMARY KEY will create implicit index "partition_table_1_pkey" for table "partition_table_1"
```

通过 sql 语句可以看出，这种方法利用数据上边界将创建分区表。而为了保证区间的完整性，应该按照最大值升序的方式创建分区。但这种方式的弊端就是上边界的类型需要限制，如数值型等等

通过 select 查看各分区表信息

```
--------------开始执行--------------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
select relname,parttype,parentid,boundaries from pg_partition where parentid in(select "oid" from pg_class
where "relname"='partition_table_1');
执行成功，当前返回：[50]行，耗时：[13ms.]
```




(2) 方法二： start end

```
---------------开始执行---------------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
create table partition_table_2(
id serial primary key,
col varchar(8))
partition by range(id)
(
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)
);
执行成功，耗时：[65ms.]

warning:
CREATE TABLE will create implicit sequence "partition_table_2_id_seq" for serial column "partition_table_2.id"
```

Every 语句将 2 到 100 之间以间隔为 10 分成了 16 个区间，2-100 有 11 个，200、300、400、500、600 总共加起来有 16 个，加上 partition_table_2 返回信息总共有 17 条

通过 select 语句查看分区信息

```
---------------开始执行---------------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
select relname , parttype , parentid, boundaries from pg_partition where parentid in (
        SELECT oid from pg_class where relname = 'partition_table_2' and relnamespace = 16916)
执行成功，当前返回：[17]行，耗时：[7ms.]
```

| | relname | parttype | parentid | boundaries |
|---|---|---|---|---|
| 11 | p1_5 | p | 36448 | {52} |
| 12 | p1_4 | p | 36448 | {42} |
| 13 | p1_3 | p | 36448 | {32} |
| 14 | p1_2 | p | 36448 | {22} |
| 15 | p1_1 | p | 36448 | {12} |
| 16 | p1_0 | p | 36448 | {2} |
| 17 | partition_table_2 | r | 36448 | |

可能多人共用一个数据库的原因，查询分区信息的时候也能查询到他人的分区信息，导致无法分区哪一些信息是自己的，所以通过观察 pg_class 的字段后，发现 relnamespace 可能是区分数据的一个字段，而通过查看表的索引发现 16916 这个数字，故猜测这个就是 relnamespace。将查询语句添加另一个约束 relnamespace = “16916”后就可以只显示自己的分区信息了。

(3) 方法三 interval

```
---------------开始执行---------------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
create table partition_table_3(
id serial,
col timestamptz)
partition by range(col)
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')
);
执行成功，耗时：[12ms.]
warning:
CREATE TABLE will create implicit sequence "partition_table_3_id_seq" for serial column "partition_table_3.id"
```

通过 select 语句查看分区信息

```
select relname , parttype , parentid, boundaries from pg_partition where parentid in (
    SELECT oid from pg_class where relname = 'partition_table_3')
```

```
---------------开始执行---------------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
select relname , parttype , parentid, boundaries from pg_partition where parentid in (
    SELECT oid from pg_class where relname = 'partition_table_3')
执行成功，当前返回：[31]行，耗时：[7ms.]
```

| | | | | |
|---|---|---|---|---|
| 29 | partition_table_3 | r | 36497 | |
| 30 | p1 | p | 36497 | {"2021-03-08 00:00:00+08"} |
| 31 | p2 | p | 36497 | {"2021-03-09 00:00:00+08"} |

可以看出这种方式通过设置数据间的间隔的方式分区。不在同一个间隔区间内的数据也将被分配到两个不同的分区表中

插入两条数据

```
---------------开始执行---------------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
insert into partition_table_3 values (1,'2021-03-11 00:00:00');
执行成功，耗时：[10ms.]
```

```
---------------开始执行---------------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
insert into partition_table_3 values(1,'2022-01-07 00:00:00');
执行成功，耗时：[9ms.]
```

再次查看分区表信息

select relname , parttype , parentid, boundaries from pg_partition where parentid in (

SELECT oid from pg_class where relname = 'partition_table_3' and relnamespace = 16916)

以下是select relname , parttype , parentid, boundaries from pg_partition where pare...的... ⓘ 该表不可编辑。 复制行 复制列 列设置

| | relname | parttype | parentid | boundaries |
|---|---|---|---|---|
| 1 | sys_p2 | p | 36497 | {"2022-01-08 00:00:00+08"} |
| 2 | sys_p1 | p | 36497 | {"2021-03-12 00:00:00+08"} |
| 3 | p2 | p | 36497 | {"2021-03-09 00:00:00+08"} |
| 4 | p1 | p | 36497 | {"2021-03-08 00:00:00+08"} |
| 5 | partition_table_3 | r | 36497 | |

可见数据库会根据 interval 的间隔自动为新增的元素添加分区

## 2.4 管理分区表

(1) 删除分区

alter table partition_table_1 drop partition p1;

```
---------------开始执行---------------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
alter table partition_table_1 drop partition p1;
执行成功，耗时：[10ms.]
```

用 select 语句查看分区

select relname , parttype , parentid, boundaries from pg_partition where parentid in (

SELECT oid from pg_class where relname = 'partition_table_1' and relnamespace = 16916)

| | relname | parttype | parentid | boundaries |
|---|---|---|---|---|
| 1 | p4 | p | 36184 | {NULL} |
| 2 | p3 | p | 36184 | {300} |
| 3 | p2 | p | 36184 | {200} |
| 4 | partition_table_1 | r | 36184 | |

(2) 添加分区

```
1 alter table partition_table_1 add partition p_1 VALUES less than(100)
```

SQL执行记录 消息

```
----------开始执行----------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
alter table partition_table_1 add partition p_1 VALUES less than(100)
执行失败，失败原因：ERROR: upper boundary of adding partition MUST overtop last existing partition
```

添加失败，应从最后开始添加，所以还应删除最后 maxvalue 的分区再继续添加

```
----------开始执行----------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
alter table partition_table_1 drop partition p4
执行成功，耗时：[9ms.]
```

```
----------开始执行----------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
alter table partition_table_1 add partition p4_new VALUES less than (500)
执行成功，耗时：[13ms.]
```

查看分区表信息

以下是select relname , parttype , parentid, boundaries from pg_partition where pare...的... ⓘ该表不可编辑。 复制行 复制列 列设置

| | relname | parttype | parentid | boundaries |
|---|---|---|---|---|
| 1 | p4_new | p | 36184 | {500} |
| 2 | p3 | p | 36184 | {300} |
| 3 | p2 | p | 36184 | {200} |
| 4 | partition_table_1 | r | 36184 | |

(3) 重命名分区表名字

```
----------开始执行----------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
alter table partition_table_1 rename partition p4_new to p4_new_new
执行成功，耗时：[11ms.]
```

查看分区表信息

| | relname | parttype | parentid | boundaries |
|---|---|---|---|---|
| 1 | p4_new_new | p | 36184 | {500} |
| 2 | p3 | p | 36184 | {300} |
| 3 | p2 | p | 36184 | {200} |
| 4 | partition_table_1 | r | 36184 | |

(4) 查询分区表情况

select relname , parttype , parentid, boundaries from pg_partition where parentid in (

SELECT oid from pg_class where relname = 'partition_table_1' and relnamespace = 16916)

查询单独分区内的数据

查询刚刚插入的两条数据，sys_p2 由系统自动生成

SELECT * FROM partition_table_3 PARTITION(sys_p2);

```
【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
SELECT * FROM partition_table_3 PARTITION(sys_p2);
执行成功，当前返回：[1]行，耗时：[7ms.]
```

以下是SELECT * FROM partition_table_3 PARTITION(sys_p2);的执行结果集　　① 该表不可编辑。　　复制行　复制列　列设置

| | id | col |
|---|---|---|
| 1 | 1 | 2022-01-07 00:00:00+08 |

(5) 数据转移

创建一个与 partition_table_3 数据格式一样的普通表

create table normal_table_3(

id serial,

col TIMESTAMPTZ )

```
【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
create table normal_table_3(
id serial,
col TIMESTAMPTZ )
执行成功，耗时：[19ms.]

warning:
CREATE TABLE will create implicit sequence "normal_table_3_id_seq" for serial column "normal_table_3.id"
```

向普通表中插入数据

INSERT INTO normal_table_3 VALUES(1,'2021-03-23 00:00:00')

```
SELECT * from normal_table_3
```

SQL执行记录 消息 结果集1 ×

以下是SELECT * from normal_table_3的执行结果集 ⓘ该表不可编辑。 复制行 复制列 列设置

覆盖模式

| | id | col |
|---|---|---|
| 1 | 1 | 2021-03-11 00:00:00+08 |
| 2 | 1 | 2021-03-12 00:00:00+08 |
| 3 | 1 | 2021-03-13 00:00:00+08 |
| 4 | 1 | 2021-03-23 00:00:00+08 |

将普通表的数据转移到分区表中

INSERT INTO partition_table_3 SELECT * from normal_table_3

```
--------------开始执行--------------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
INSERT INTO partition_table_3 SELECT * from normal_table_3
执行成功，耗时：[12ms.]
```

查看分区表分区情况

以下是select relname , parttype , parentid, boundaries from pg_partition where pare...的... ⓘ该表不可编辑。 复制行 复制列 列设置

| | relname | parttype | parentid | boundaries |
|---|---|---|---|---|
| 1 | sys_p5 | p | 36497 | {"2021-03-24 00:00:00+08"} |
| 2 | sys_p4 | p | 36497 | {"2021-03-14 00:00:00+08"} |
| 3 | sys_p3 | p | 36497 | {"2021-03-13 00:00:00+08"} |
| 4 | sys_p2 | p | 36497 | {"2022-01-08 00:00:00+08"} |
| 5 | sys_p1 | p | 36497 | {"2021-03-12 00:00:00+08"} |
| 6 | p2 | p | 36497 | {"2021-03-09 00:00:00+08"} |
| 7 | p1 | p | 36497 | {"2021-03-08 00:00:00+08"} |

自动添加了多个分区

查询某个分区的内容

SELECT * from partition_table_3 PARTITION (sys_p5)

| | id | col |
|---|---|---|
| 1 | 1 | 2021-03-23 00:00:00+08 |

# 三、 索引

## 3.3 普通表上创建管理索引

1. 创建一个普通表

```sql
CREATE table table_0(
  num1 integer not null,
  num2 INTEGER not null,
  num3 integer not null,
  num4 INTEGER not null,
  num5 INTEGER not null)
```

SQL执行记录 消息

```
---------------开始执行---------------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
CREATE table table_0(
  num1 integer not null,
  num2 INTEGER not null,
  num3 integer not null,
  num4 INTEGER not null,
  num5 INTEGER not null)
执行成功，耗时：[10ms.]
```

2. 创建普通索引

在 num1 上创建一个普通索引

```sql
create index table_0_index_num1 on table_0(num1);
```

SQL执行记录 消息

```
---------------开始执行---------------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
create index table_0_index_num1 on table_0(num1);
执行成功，耗时：[15ms.]
```

3. 创建多字段索引

在 nun2，num3 上创建多字段索引

```sql
CREATE INDEX table_0_index_more_column ON table_0(num2,num3);
```

SQL执行记录 消息

```
---------------开始执行---------------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
CREATE INDEX table_0_index_more_column ON table_0(num2,num3);
执行成功，耗时：[13ms.]
```

4. 仅对数据的某一部分创建索引

```
1 CREATE INDEX table_0_part_index ON table_0(num4) WHERE num4=2;
```

SQL执行记录 消息

```
----------------开始执行----------------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
CREATE INDEX table_0_part_index ON table_0(num4) WHERE num4=2;
执行成功，耗时：[12ms.]
```

5. 创建表达式索引

使用场景：经常需要查询 num5 的平方的信息

```
1 CREATE INDEX table_0_para_index ON table_0(power(num5,2));
```

SQL执行记录 消息

```
----------------开始执行----------------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
CREATE INDEX table_0_para_index ON table_0(power(num5,2));
执行成功，耗时：[12ms.]
```

6. 管理索引

查询语句：

SELECT relname FROM PG_CLASS WHERE RELKIND='i' and relnamespace = 16916;

SQL执行记录 消息 结果集1 X

以下是SELECT relname FROM PG_CLASS WHERE RELKIND='i' and relnamespace = ... ⓘ该表不可编辑。

| | relname |
|---|---|
| 1 | partition_table_1_pkey |
| 2 | partition_table_2_pkey |
| 3 | pk_sector_id |
| 4 | table_0_index_more_column |
| 5 | table_0_index_num1 |
| 6 | table_0_para_index |
| 7 | table_0_part_index |
| 8 | tbATUData_pkey |

就可以看到刚刚建立的索引

7. 重命名索引

```
1 ALTER INDEX table_0_index_more_column RENAME TO table_0_index_num2_num3;
```

SQL执行记录 消息

----------开始执行----------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
ALTER INDEX table_0_index_more_column RENAME TO table_0_index_num2_num3;
执行成功，耗时：[13ms.]

执行再次执行索引查询语句

```
1 SELECT relname FROM PG_CLASS WHERE RELKIND='i' and relnamespace = 16916;
```

SQL执行记录 消息 结果集1 X

以下是SELECT relname FROM PG_CLASS WHERE RELKIND='i' and relnamespace = ... ⓘ该表不可编辑。

| | relname |
|---|---|
| 1 | partition_table_1_pkey |
| 2 | partition_table_2_pkey |
| 3 | pk_sector_id |
| 4 | table_0_index_num1 |
| 5 | table_0_index_num2_num3 |
| 6 | table_0_para_index |
| 7 | table_0_part_index |
| 8 | tbATUData_pkey |

第四条数据中，名字已经改成了 table_0_index_num2_num3

8. 删除索引

删除之前：

```
SELECT relname FROM PG_CLASS WHERE RELKIND='i' and relnamespace = 16916 and relname like 'table_0%';
```

SQL执行记录 消息 结果集1 ×

以下是SELECT relname FROM PG_CLASS WHERE RELKIND='i' and relnamespace = ... ⓘ该表不可编辑。

| | relname |
|---|---|
| 1 | table_0_index_num1 |
| 2 | table_0_index_num2_num3 |
| 3 | table_0_part_index |

执行删除语句

```
drop index table_0_part_index;
```

SQL执行记录 消息

```
----------开始执行----------
【拆分SQL完成】：将执行SQL语句数量：（1条）
【执行SQL：(1)】
drop index table_0_part_index;
执行成功，耗时：[9ms.]
```

再次查询

```
SELECT relname FROM PG_CLASS WHERE RELKIND='i' and relnamespace = 16916 and relname like 'table_0%';
```

SQL执行记录 消息 结果集1 ×

以下是SELECT relname FROM PG_CLASS WHERE RELKIND='i' and relnamespace = ... ⓘ该表不可编辑。

| | relname |
|---|---|
| 1 | table_0_index_num1 |
| 2 | table_0_index_num2_num3 |

可见对应索引被删除了

## 3.4 在分区表上创建管理索引

1. 创建一个分区表

```
---------------开始执行---------------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
create table partition_table_0(
id serial primary key,
num1 integer,
num2 integer,
num3 integer)
partition by range(id)
(
partition p1 values less than(100),
partition p2 values less than(200),
partition p3 values less than(300),
partition p4 values less than(maxvalue)
);
执行成功，耗时：[25ms.]
```

2. 创建 global 索引

```
1 create index global_index_num1 on partition_table_0(num1) global
```

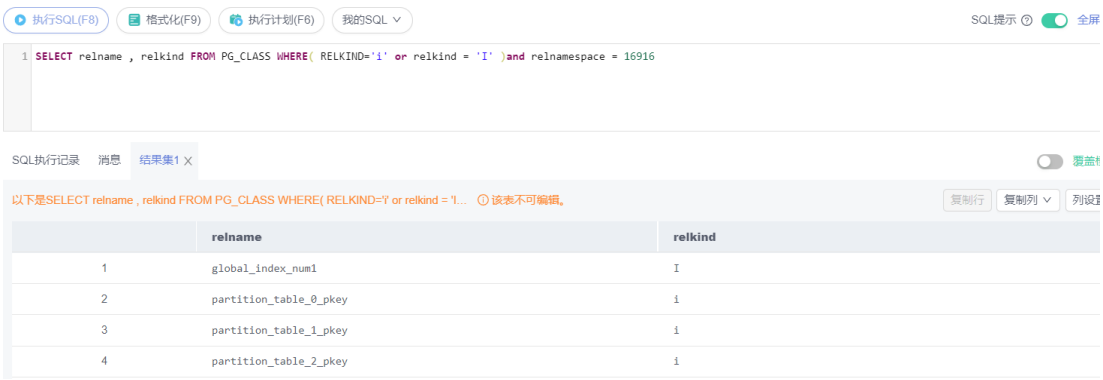
SQL执行记录 消息

```
---------------开始执行---------------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
create index global_index_num1 on partition_table_0(num1) global
执行成功，耗时：[15ms.]
```

有关表空间的语句被省略

通过查询语句

SELECT relname , relkind FROM PG_CLASS WHERE( RELKIND='i' or relkind = 'I' )and relnamespace = 16916




可见索引已经创建成功，且类别为 global 索引

3. 创建不指定分区的 local 索引

```
1 create index local_index_num2 on partition_table_0 (num2) local
```

SQL执行记录 消息

```
---------------开始执行---------------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
create index local_index_num2 on partition_table_0 (num2) local
执行成功，耗时：[25ms.]
```

4. 创建指定索引分区名称的 local 索引

```
1 create index local_index_num3 on partition_table_0(num3) local
2 (
3 partition p1_index,
4 partition p2_index,
5 partition p3_index ,
6 partition p4_index
7 ) ;
```

SQL执行记录 消息

```
---------------开始执行---------------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
create index local_index_num3 on partition_table_0(num3) local
(
partition p1_index,
partition p2_index,
partition p3_index ,
partition p4_index
) ;
执行成功，耗时：[24ms.]
```

5. 管理索引

(1) 修改索引分区所在表空间

略

(2) 重命名索引分区

```sql
ALTER INDEX local_index_num3 RENAME PARTITION p2_index TO p2_index_new;
```

SQL执行记录 消息

```
---------------开始执行---------------

【拆分SQL完成】：将执行SQL语句数量：（1条）

【执行SQL：(1)】
ALTER INDEX local_index_num3 RENAME PARTITION p2_index TO p2_index_new;
执行成功，耗时：[10ms.]
```

(3) 删除索引

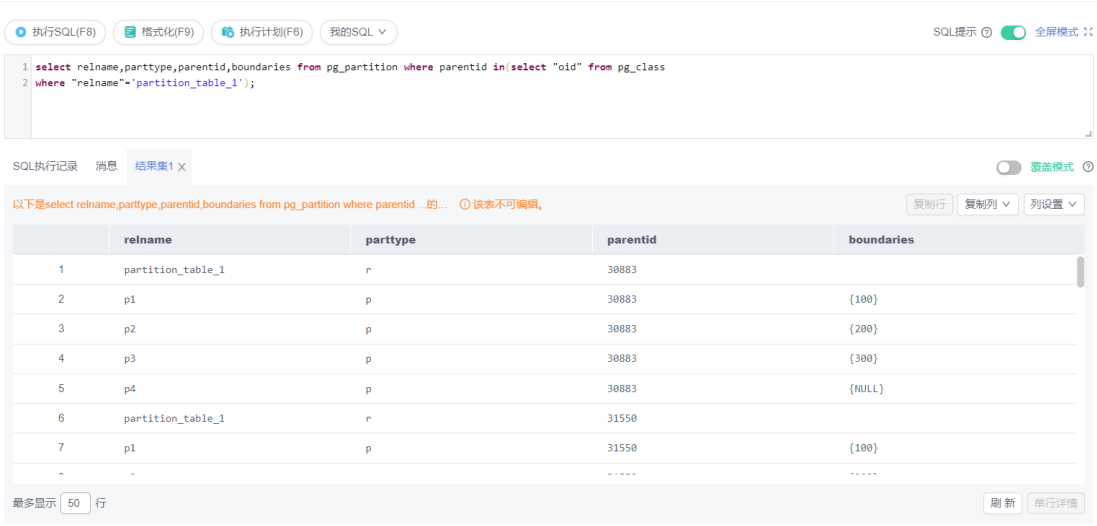
```sql
drop index global_index_num1;
drop index local_index_num2;
drop index local_index_num3;
```

SQL执行记录 消息

```
---------------开始执行---------------

【拆分SQL完成】：将执行SQL语句数量：（3条）

【执行SQL：(1)】
drop index global_index_num1;
执行成功，耗时：[13ms.]

【执行SQL：(2)】
drop index local_index_num2;
执行成功，耗时：[15ms.]

【执行SQL：(3)】
drop index local_index_num3;
执行成功，耗时：[11ms.]
```

# 四、 遇到的问题及解决

问题一：可能多人共用一个数据库的原因，查询分区信息的时候也能查询到他人的分区信息，导致无法分区哪一些信息是自己的。




如图中出现了多个 p1

问题解决：

通过观察 pg_class 的字段后，发现 relnamespace 可能是区分数据的一个字段，而通过查看表的索引发现 16916 这个数字，故猜测这个就是 relnamespace。将查询语句添加另一个约束 relnamespace = “16916”后就可以只显示自己的分区信息了。

```
select relname , parttype , parentid, boundaries from pg_partition where parentid in (
 SELECT oid from pg_class where relname = 'partition_table_3' and relnamespace = 16916)
```


# 五、 实验总结

这次实验的难度较大，主要原因为分区表等操作之前从未操作过，导致分区表的理解上存在困难。且之前的实验均与数据库的物理设计没有太多的关系，所以对数据库的物理设计这一块的知识储备较薄弱。但我们都知道一个运行良好的数据库不能仅仅依赖 sql 语句上的优化，它也需要运行在一个配置得当的物理设计上，如当数据量非常大时需要分区分表以加快搜索、在分区后的表上创建管理索引等。这些都是可预见的将来都需要用到的知识。所以

经过这一次实验，我对数据库物理设计的理解进一步加深，让我受益匪浅。