**分区表使用场景**

表分区，是指根据一定规则，将数据库中的一张表分解成多个更小的，容易管理的部分。从逻辑上看，只有一张表，但是底层却是由多个物理分区组成。

分区是一种表的设计模式，正确的分区可以极大地提升数据库的查询效率，完成更高质量的SQL编程。但是如果错误地使用分区，那么分区可能带来毁灭性的的结果。

随着使用时间的增加，数据库中的数据量也不断增加，因此数据库查询越来越慢。

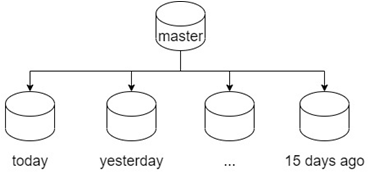
加速数据库的方法很多，如添加特定的索引，将日志目录换到单独的磁盘分区，调整数据库引擎的参数等。这些方法都能将数据库的查询性能提高到一定程度。

对于许多应用数据库来说，许多数据是历史数据并且随着时间的推移它们的重要性逐渐降低。如果能找到一个办法将这些可能不太重要的数据隐藏，数据库查询速度将会大幅提高。可以通过DELETE来达到此目的，但同时这些数据就永远不可用了。

因此，需要一个高效的把历史数据从当前查询中隐藏起来并且不造成数据丢失的方法。数据库表分区即能达到此效果。

**表分区介绍**

表分区把一个大的物理表分成若干个小的物理表，并使得这些小物理表在逻辑上可以被当成一张表来使用。



**表分区术语介绍**

主表/父表/Master Table　该表是创建子表的模板。它是一个正常的普通表，但正常情况下它并不储存任何数据。

子表/分区表/Child Table/Partition Table　这些表继承并属于一个主表。子表中存储所有的数据。主表与分区表属于一对多的关系，也就是说，一个主表包含多个分区表，而一个分区表只从属于一个主表

**表分区的优势：**

删除历史数据更快， 如果是按时间分区的， 在删除历史数据时， 直接删除历史分区即可， 如果没有分区， 通过DELETE删除历史数据时会很慢， 还容易导致VACUUM占用大量的IO。

对于某些类型的查询，尤其是当表中大多数被频繁访问的行位于单个分区或少数几个分区中时，查询性能可以得到显着提高。例如，在按时间分区的表中， 如果大多数查询发生在时间最近的一个或几个分区中， 而较早时间的分区较少查询， 那么， 在建分区表后， 各个分区表均有各自的索引， 使用率较高的分区表的索引就可能完全缓存在内存中， 这样效率就会提高很多。

当查询或更新一个分区的大部分记录时， 连续扫描该分区而不是使用索引离散地访问整个表， 可以获得巨大的性能提升。

很少用到的历史数据可以使用表空间的技术移动到便宜一些的慢速存储介质上。 因为使用分区表可以将不同的分区安置在不同的物理介质上。

如果表的数据太大，一个磁盘放不下，利用分区表可以把数据分配到不同的磁盘里面去。

以上优势只有当表非常大的时候才能体现出来。一般来说，当表的大小超过数据库服务器的物理内存时以上优势才能体现出来。

多大数据适合使用分区表？ 一般取决于具体的应用， 不过也有个简单的基本原则， 即表的大小超过了数据库服务器的物理内存大小时使用。

**PostgreSQL表分区**

PostgreSQL内部是通过表继承来实现分区表的。父表是普通表并且正常情况下并不存储任何数据，它的存在只是为了代表整个数据集，数据是存储在子表中的。PostgreSQL可实现如下三种表分区。

范围分区：partition by range(…)

该表被划分为由键列或列集定义的" 范围 "，分配给不同分区的值的范围之间没有重叠。例如，可以按日期范围或特定业务对象的标识符范围进行分区。

列表分区:partition by list(…)

通过显式列出哪些键值出现在每个分区中来对表进行分区。

哈希分区：parition by hash(…)

通过为每个分区指定模数和余数来对表进行分区。每个分区将保留行，分区键的哈希值除以指定的模数将产生指定的余数。

PostgreSQL 10 版本之前只能通过表继承来实现分区表。 而PostgreSQL 10 版本之后支持声明式分区，使用相应的DDL语句就可以直接创建分区表， 但内部原理仍是表继承。

判断当前PostgreSQL是否支持声明式分区？

select version();

show enable\_partition\_pruning;

**分区表定义语法个数**

使用create table语句定义分区表和分区：

  创建分区表：create table tablename (…) partition by (…)

  创建分区： create table partitionname partition of tablename for values (…);

partition by (…)：定义分区的列和算法

for values (…)：定义分区的取值范围

**创建继承式分区**

1.创建主表，所有的子表都从主表继承，主表不包含数据，不在主表上定义任何检查约束。

create table log\_history(id int not null,logdate date not null,info text);

2.创建子表

create table log\_history\_2013(check (logdate >= date'2013-01-01' and logdate < date'2014-01-01')) inherits(log\_history);

create table log\_history\_2014(check (logdate >= date'2014-01-01' and logdate < date'2015-01-01')) inherits(log\_history);

create table log\_history\_2015(check (logdate >= date'2015-01-01' and logdate < date'2016-01-01')) inherits(log\_history);

create table log\_history\_2016(check (logdate >= date'2016-01-01' and logdate < date'2017-01-01')) inherits(log\_history);

create table log\_history\_2017(check (logdate >= date'2017-01-01' and logdate < date'2018-01-01')) inherits(log\_history);

create table log\_history\_2018(check (logdate >= date'2018-01-01' and logdate < date'2019-01-01')) inherits(log\_history);

create table log\_history\_2019(check (logdate >= date'2019-01-01' and logdate < date'2020-01-01')) inherits(log\_history);

create table log\_history\_2020(check (logdate >= date'2020-01-01' and logdate < date'2021-01-01')) inherits(log\_history);

create table log\_history\_2021(check (logdate >= date'2021-01-01' and logdate < date'2022-01-01')) inherits(log\_history);

create table log\_history\_2022(check (logdate >= date'2022-01-01' and logdate < date'2023-01-01')) inherits(log\_history);

查看继承分区

select inhrelid::regclass,inhparent::regclass,inhseqno from pg\_inherits;

3.分区表创建后，往主表插入数据并不会将数据重定向到合适的分区，因此需要创建合适的触发器函数来实现这一点。

insert into log\_history values(1,'2021-09-01','xxx');

select \* from log\_history;--有数据

select \* from log\_history\_2021;--无数据

4.创建函数

CREATE OR REPLACE FUNCTION log\_history\_insert\_function()

RETURNS TRIGGER AS $$

BEGIN

IF ( NEW.logdate >= DATE '2013-01-01' AND NEW.logdate < DATE '2014-01-01' ) THEN

INSERT INTO log\_history\_2013 VALUES (NEW.);ELSIF ( NEW.logdate >= DATE '2014-01-01' ANDNEW.logdate < DATE '2015-01-01' ) THENINSERT INTO log\_history\_2014 VALUES (NEW.);

ELSIF ( NEW.logdate >= DATE '2015-01-01' AND NEW.logdate < DATE '2016-01-01' ) THEN

INSERT INTO log\_history\_2015 VALUES (NEW.);ELSIF ( NEW.logdate >= DATE '2016-01-01' ANDNEW.logdate < DATE '2017-01-01' ) THENINSERT INTO log\_history\_2016 VALUES (NEW.);

ELSIF ( NEW.logdate >= DATE '2017-01-01' AND NEW.logdate < DATE '2018-01-01' ) THEN

INSERT INTO log\_history\_2017 VALUES (NEW.);ELSIF ( NEW.logdate >= DATE '2018-01-01' ANDNEW.logdate < DATE '2019-01-01' ) THENINSERT INTO log\_history\_2018 VALUES (NEW.);

ELSIF ( NEW.logdate >= DATE '2019-01-01' AND NEW.logdate < DATE '2020-01-01' ) THEN

INSERT INTO log\_history\_2019 VALUES (NEW.);ELSIF ( NEW.logdate >= DATE '2020-01-01' ANDNEW.logdate < DATE '2021-01-01' ) THENINSERT INTO log\_history\_2020 VALUES (NEW.);

ELSIF ( NEW.logdate >= DATE '2021-01-01' AND NEW.logdate < DATE '2022-01-01' ) THEN

INSERT INTO log\_history\_2021 VALUES (NEW.);ELSIF ( NEW.logdate >= DATE '2022-01-01' ANDNEW.logdate < DATE '2023-01-01' ) THENINSERT INTO log\_history\_2022 VALUES (NEW.);

ELSE

RAISE EXCEPTION 'Date out of range. Fix the log\_history\_insert\_trigger() function!';

END IF;

RETURN NULL;

END;

$$

LANGUAGE plpgsql;

5.创建触发器

CREATE TRIGGER log\_history\_insert\_trigger

BEFORE INSERT ON log\_history

FOR EACH ROW EXECUTE PROCEDURE log\_history\_insert\_function();

注意：分区表中只包括了2013年1月到2022年12月的分区表， 如果插入其他日期的数据会报错，需要添加新的分区表，同时修改触发器的内容。

insert into log\_history select random()\*120,n,md5(random()::text) from generate\_series('2013-01-01'::date,'2022-12-31'::date,'1 month') n;

6.创建索引（子表不会继承主表的索引）

create index index\_log\_history\_logdate on log\_history(logdate);

create index index\_log\_history\_2013\_logdate on log\_history\_2013(logdate);

create index index\_log\_history\_2014\_logdate on log\_history\_2014(logdate);

create index index\_log\_history\_2015\_logdate on log\_history\_2015(logdate);

create index index\_log\_history\_2016\_logdate on log\_history\_2016(logdate);

create index index\_log\_history\_2017\_logdate on log\_history\_2017(logdate);

create index index\_log\_history\_2018\_logdate on log\_history\_2018(logdate);

create index index\_log\_history\_2019\_logdate on log\_history\_2019(logdate);

create index index\_log\_history\_2020\_logdate on log\_history\_2020(logdate);

create index index\_log\_history\_2021\_logdate on log\_history\_2021(logdate);

create index index\_log\_history\_2022\_logdate on log\_history\_2022(logdate);

**分区优化技巧**

constraint\_exclusion参数可以提升声明式分区表性能的查询优化技术。

select \* from log\_history where logdate = date'2020-09-01';

如果启用constraint\_exclusion参数，上面的查询将会扫描log\_history表的每一个分区。如果启用了此参数，规划器将会检查每个分区的定义并且检验该分区是否因为不包含符合查询WHERE子句的行而无需扫描。当规划器可以证实这一点时，它会把分区从查询计划中排除。

启用constraint\_exclusion = partition参数（默认启用partition），指定计划只会扫描需要的分区

set constraint\_exclusion = partition;

explain select \* from log\_history where logdate=date'2020-09-01';

如果未启用constraint\_exclusion参数，指定计划只会扫描需要的分区

set constraint\_exclusion = off;

explain select \* from log\_history where logdate=date'2020-09-01';

**声明式分区**

**声明式分区的特点**

在分区表上建立索引，相应的所有分区都能自动建立索引，以后创建或附加的任何分区也将包含该索引。也可以为分区单独建立索引。

可以根据需要，卸载或增加一个分区。

分区可以指定单独的表空间，能充分利用多个磁盘。

分区可以指向一个PG外表，即FDW表。

分区表可以根据多个列的值来分区。

分区可以再次分区。

**范围分区**

根据范围分区，范围应该连续但是不重叠，使用PARTITION BY RANGE, FOR VALUES FROM关键字。

RANGE分区适用分区键为日期或者时间类型，数据分布均衡，容易归档。

**根据时间范围**

当往分区列中插入null值RANG 分区会将其当作最小值来处理即插入最小的分区中

1.创建分区表并指定分区方式

create table log\_history(id int not null,logdate date not null,info text) partition by range(logdate);

2.创建分区

create table log\_history\_2018 partition of log\_history for values from ('2018-01-01') to ('2019-01-01');

create table log\_history\_2019 partition of log\_history for values from ('2019-01-01') to ('2020-01-01');

create table log\_history\_2020 partition of log\_history for values from ('2020-01-01') to ('2021-01-01');

create table log\_history\_2021 partition of log\_history for values from ('2021-01-01') to ('2022-01-01');

create table log\_history\_2022 partition of log\_history for values from ('2022-01-01') to ('2023-01-01');

注意：前后时间范围分配衔接

查看分区

\d+ log\_history;

\d+ log\_history\_2022;

insert into log\_history values(1, '2018-05-20','success');

insert into log\_history values(2, '2019-07-03','success');

insert into log\_history values(3, '2020-02-06','fail');

insert into log\_history values(4, '2021-09-31','success');

insert into log\_history values(5, '2022-02-14','success');

insert into log\_history values(6, '2023-11-22','fail');--ERROR

create table log\_history\_default partition of log\_history default;--默认分区

insert into log\_history values(6, '2023-11-22','fail');--ERROR

**根据数值范围分区**

创建一张表带年龄的主表，然后根据年龄分段来进行分区：

CREATE TABLE person (

name varchar not null,

age int not null,

city varchar not null

) PARTITION BY RANGE (age);

创建四张分区表，分别对应年龄是11到20岁、21到30岁、31到40岁、41岁到50岁。

create table person\_p1 partition of person for values from (11) to (21);

create table person\_p2 partition of person for values from (21) to (31);

create table person\_p3 partition of person for values from (31) to (41);

create table person\_p4 partition of person for values from (41) to (51);

插入测试数据：

insert into person(age, city) VALUES ('wangwu',21, 'SZ');

insert into person(age, city) VALUES ('zhaoliu',13, 'BJ');

insert into person\_(age, city) VALUES ('zhouba',28, 'HK');

insert into person(age, city) VALUES ('sunqi',43, 'SH');

insert into person(age, city) VALUES ('lisi',2, 'SZ'); --ERROR

create table person\_p0 partition of person for values from (MINVALUE) to (11);

create table person\_p5 partition of person for values from (51) to (MAXVALUE);

**指定分区表空间**

-- 创建主表

create table student(

id int not null,

name varchar,

age int

)partition by range(id);

-- 创建分区表

create table student\_p0 partition of student for values from (1) to (20001) tablespace ts\_p0;

create table student\_p1 partition of student for values from (20001) to (40001) tablespace ts\_p1;

create table student\_p2 partition of student for values from (40001) to (60001) tablespace ts\_p2;

create table student\_p3 partition of student for values from (60001) to (80001) tablespace ts\_p3;

create table student\_p4 partition of student for values from (80001) to (100001) tablespace ts\_p4;

-- 插入测试数据

insert into student(id,name,age) select n,n\_||'user',random()\*10+20 from generate\_series(1,100000);

**列表分区**

列表分区是按特定的值来分区，每个分区值不重叠，使用PARTITION BY LIST、FOR VALUES IN关键字。

类似于按RANGE分区，区别在于LIST分区是基于列值匹配一个离散值集合中的某个值来进行选择。

**根据字符串列表分区：**

假如以地区列表来进行分区

-- 创建主表

create table person (

name varchar not null,

age int not null,

city varchar not null

) partition by list (city);

-- 创建分区表

CREATE TABLE person\_p1 PARTITION OF person FOR VALUES IN ('sc','yn','gz');

CREATE TABLE person\_p2 PARTITION OF person FOR VALUES IN ('gd','gx');

CREATE TABLE person\_p3 PARTITION OF person FOR VALUES IN ('bj','sh','sz');

CREATE TABLE person\_p4 PARTITION OF person DEFAULT;

-- 插入测试数据

insert into person VALUES ('zhangsan',22,'sc');

insert into person VALUES ('lisi',18,'gx');

insert into person VALUES ('wangwu',20,'bj');

insert into person VALUES ('zhaoliu',24,'SC');

**根据数值列表分区：**
假定有20个音像店，分布在4个有经销权的地区，如下表所示：

地区      商店ID号

北区      3, 5, 6, 9, 17

东区      1, 2, 10, 11, 19, 20

西区      4, 12, 13, 14, 18

中心区  7, 8, 15, 16

create table staff(

id int not null,

name varchar(30),

age int,

shop\_id int not null default 0

)partition by list(id);

CREATE TABLE staff\_north PARTITION OF staff FOR VALUES IN (3,5,6,9,17);

CREATE TABLE staff\_east PARTITION OF staff FOR VALUES IN (1,2,10,11,19,20);

CREATE TABLE staff\_west PARTITION OF staff FOR VALUES IN (4,12,13,14,18);

CREATE TABLE staff\_center PARTITION OF staff FOR VALUES IN (7,8,15,16);

这使得在表中增加或删除指定地区的员工记录变得容易起来。例如，假定西区的所有音像店都卖给了其他公司。那么与在西区音像店工作员工相关的所有记录（行）可以使用“drop table staff\_west；”来进行删除，它与具有同样作用的DELETE（删除）“DELETE FROM staff WHERE shop\_id IN (4,12,13,14,18)；”比起来，要有效得多。

**哈希分区**

哈希分区主要用来确保数据在预先确定数目的分区中平均分布，Hash分区支几乎支持的所有数据类型。

要使用HASH分区来分割一个表，要在CREATE TABLE 语句上添加一个“PARTITION BY HASH”子句和“for values with”。

哈希分区是指按字段取哈希值后再分区。

**根据数值类型来分区**

-- 创建主表

create table person (

name varchar not null,

age int not null,

city text not null

) partition by hash (age);

-- 创建分区表

create table person\_p0 partition of person for values with (modulus 4, remainder 0);

create table person\_p1 partition of person for values with (modulus 4, remainder 1);

create table person\_p2 partition of person for values with (modulus 4, remainder 2);

create table person\_p3 partition of person for values with (modulus 4, remainder 3);

insert into person select substring(md5(n::text),1,5),random()\*100, md5(n::text) from generate\_series(1,10000) n;

select (hashint4extended(int4, 8816678312871386365)::numeric + 5305509591434766563) %32;--hash算法

**根据字符类型分区**

-- 创建主表

create table person (

name varchar not null,

age int not null,

city text not null

) partition by hash (city);

-- 创建分区表

create table person\_p0 partition of person for values with (modulus 4, remainder 0);

create table person\_p1 partition of person for values with (modulus 4, remainder 1);

create table person\_p2 partition of person for values with (modulus 4, remainder 2);

create table person\_p3 partition of person for values with (modulus 4, remainder 3);

-- 插入测试数据

insert into person select md5(n::text),random()\*100,substring(md5(n::text),1,5) from generate\_series(1,10000) n;

SELECT (hashtextextended('value', 8816678312871386365)::numeric + 5305509591434766563) % 8;--hash算法

**组合分区**

组合分区可以结合range、list和hash，根据业务需要提供灵活的玩法。

**range与hash组合**

-- 创建range分区主表

create table student(

id int not null,

name varchar(255),

age int not null

) partition by range(id);

--创建二级分区主表

create table student\_r1 partition of student for values from (1) to (10001) partition by hash(age);

create table student\_r2 partition of student for values from (10001) to (20001) partition by hash(age);

create table student\_r3 partition of student for values from (20001) to (30001) partition by hash(age);

--创建分区

create table student\_r1\_h0 partition of student\_r1 for values with (modulus 4, remainder 0);

create table student\_r1\_h1 partition of student\_r1 for values with (modulus 4, remainder 1);

create table student\_r1\_h2 partition of student\_r1 for values with (modulus 4, remainder 2);

create table student\_r1\_h3 partition of student\_r1 for values with (modulus 4, remainder 3);

create table student\_r2\_h0 partition of student\_r2 for values with (modulus 4, remainder 0);

create table student\_r2\_h1 partition of student\_r2 for values with (modulus 4, remainder 1);

create table student\_r2\_h2 partition of student\_r2 for values with (modulus 4, remainder 2);

create table student\_r2\_h3 partition of student\_r2 for values with (modulus 4, remainder 3);

create table student\_r3\_h0 partition of student\_r3 for values with (modulus 4, remainder 0);

create table student\_r3\_h1 partition of student\_r3 for values with (modulus 4, remainder 1);

create table student\_r3\_h2 partition of student\_r3 for values with (modulus 4, remainder 2);

create table student\_r3\_h3 partition of student\_r3 for values with (modulus 4, remainder 3);

-- 插入数据

insert into student select n,n||'\_user',random()\*100 from generate\_series(1,30000) n;

select \* from student;

**list与range组合**

-- 创建list分区主表

create table student(

id int not null,

name varchar(255),

age int not null,

city text not null

) partition by list(city);

--创建二级分区主表

create table student\_l1 partition of student for values in ('yn','gz','sc') partition by range(id);

create table student\_l2 partition of student for values in ('gd','gx') partition by range(id);

create table student\_l3 partition of student for values in ('bj','sh') partition by range(id);

--创建分区

create table student\_l1\_r1 partition of student\_l1 for values from (1) to (10001);

create table student\_l1\_r2 partition of student\_l1 for values from (10001) to (20001);

create table student\_l1\_r3 partition of student\_l1 for values from (20001) to (30001);

create table student\_l2\_r1 partition of student\_l2 for values from (1) to (10001);

create table student\_l2\_r2 partition of student\_l2 for values from (10001) to (20001);

create table student\_l2\_r3 partition of student\_l2 for values from (20001) to (30001);

create table student\_l3\_r1 partition of student\_l3 for values from (1) to (10001);

create table student\_l3\_r2 partition of student\_l3 for values from (10001) to (20001);

create table student\_l3\_r3 partition of student\_l3 for values from (20001) to (30001);

-- 插入数据

insert into student values(1,'zhangsan',22,'sc');

insert into student values(15000,'lisi',18,'gx');

insert into student values(22000,'wangwu',20,'sh');

select \* from student;

**hash与list组合**

-- 创建hash分区主表

create table student(

id int not null,

name varchar(255),

age int not null,

city text not null

) partition by hash(id);

--创建二级分区主表

create table student\_h1 partition of student for values with (modulus 3, remainder 0) partition by list(city);

create table student\_h2 partition of student for values with (modulus 3, remainder 1) partition by list(city);

create table student\_h3 partition of student for values with (modulus 3, remainder 2) partition by list(city);

--创建分区

create table student\_h1\_l1 partition of student\_h1 for values in ('yn','gz','sc');

create table student\_h1\_l2 partition of student\_h1 for values in ('gd','gx');

create table student\_h1\_l3 partition of student\_h1 for values in ('bj','sh');

create table student\_h2\_l1 partition of student\_h2 for values in ('yn','gz','sc');

create table student\_h2\_l2 partition of student\_h2 for values in ('gd','gx');

create table student\_h2\_l3 partition of student\_h2 for values in ('bj','sh');

create table student\_h3\_l1 partition of student\_h3 for values in ('yn','gz','sc');

create table student\_h3\_l2 partition of student\_h3 for values in ('gd','gx');

create table student\_h3\_l3 partition of student\_h3 for values in ('bj','sh');

-- 插入数据

insert into student values(1,'zhangsan',22,'sc');

insert into student values(15000,'lisi',18,'gx');

insert into student values(22000,'wangwu',20,'sh');

select \* from student;

**指定分区表空间**

-- 创建主表

create table student(

id int not null,

name varchar,

age int

)partition by range(id);

-- 创建分区表

create table student\_p0 partition of student for values from (1) to (20001) tablespace ts\_p0;

create table student\_p1 partition of student for values from (20001) to (40001) tablespace ts\_p1;

create table student\_p2 partition of student for values from (40001) to (60001) tablespace ts\_p2;

create table student\_p3 partition of student for values from (60001) to (80001) tablespace ts\_p3;

create table student\_p4 partition of student for values from (80001) to (100001) tablespace ts\_p4;

-- 插入测试数据

insert into student(id,name,age) select n,n\_||'user',random()\*10+20 from generate\_series(1,100000);

**分区表的管理与维护**

**查看分区表定义**

\d+ log\_history;

\d+ log\_history\_2022;

**查看分区**

查看有哪些分区表（不查看子分区）

select partrelid::regclass,\* from pg\_partitioned\_table;

\dP

**普通表转换为分区表**

方法一：

alter table student add check (id <= 10000);

alter table student\_all ATTACH PARTITION student for values from (minvalues) to (10001);

方法二：

create table student\_new(like student including defaults including constraints) partition by hash(id);

create table student\_p0 partition of student\_new for values with (modulus 5, remainder 0);

create table student\_p1 partition of student\_new for values with (modulus 5, remainder 1);

create table student\_p2 partition of student\_new for values with (modulus 5, remainder 2);

create table student\_p3 partition of student\_new for values with (modulus 5, remainder 3);

create table student\_p4 partition of student\_new for values with (modulus 5, remainder 4);

insert into student\_new select \* from student;

**分区表操作**

**1.增加分区**

create table log\_history\_2023 partition of log\_history for values from ('2023-01-01') to ('2024-01-01');

**2.删除分区**

直接删除子表删除分区

drop table log\_history\_2022;

提示：删除主表也会删除所包含的分区。

使用DETACH将分区从分区表中移除，成为一个独立的表

alter table log\_history DETACH PARTITION log\_history\_2021;

drop table log\_history\_2021;

**3.截断分区**

truncate student\_p0;

**4.数据转移**

分区键之间的数据转移(PG11后支持)

update log\_history set logdate='2022-01-15' where logdate='2020-01-01';

select \* from log\_history\_2022;

默认分区数据转移

update log\_history set logdate='2023-09-15' where logdate='2015-01-01';

select \* from log\_history\_default;

**5.VACUUM或ANALYZE**

VACUUM或ANALYZE 分区表会对其子表起作用。

查看分区表最后一次VACUUM和ANALYZE时间

select relname,last\_vacuum,last\_analyze from pg\_stat\_all\_tables where relname like 'log\_history%';

vacuum主表

vacuum log\_history;

select relname,last\_vacuum,last\_analyze from pg\_stat\_all\_tables where relname like 'log\_history%';

analyze主表

analyze log\_history;

select relname,last\_vacuum,last\_analyze from pg\_stat\_all\_tables where relname like 'log\_history%';

**分区优化技巧**

enable\_partition\_pruning参数可以提升声明式分区表性能的查询优化技术。

select \* from log\_history where logdate=date'2022-01-01';

如果启用enable\_partition\_pruning参数，上面的查询将会扫描log\_history表的每一个分区。如果启用了此参数，规划器将会检查每个分区的定义并且检验该分区是否因为不包含符合查询WHERE子句的行而无需扫描。当规划器可以证实这一点时，它会把分区从查询计划中排除。

启用enable\_partition\_pruning（默认启用此参数），查看执行计划，只扫描需要用到的分区

set enable\_partition\_pruning = on;

explain select \* from log\_history where logdate=date'2022-01-01';

未启用enable\_partition\_pruning，查看执行计划，会扫描全部分区

set enable\_partition\_pruning = off;

explain select \* from log\_history where logdate=date'2022-01-01';