

All important but hidden information about your table and table data in PostgreSQL

Presented By :-

Sachin Kotwal

Rajkumar Raghuwanshi

Agenda

- Introduction to PostgreSQL System Columns
- Data Storage and File System Representation
- Beyond the * data from the table
- Understanding Key System Columns in detail
 - ctid
 - tableoid
 - xmin and xmax
 - cmin and cmax

Understanding System Columns

```
[postgres=# create table nocol();
CREATE TABLE
[postgres=# insert into nocol default values;
INSERT 0 1
[postgres=# select attname, attnum, atttypid::regtype, attisdropped::text
[postgres=# from pg_attribute where attrelid = 'nocol'::regclass;
  attname  | attnum | atttypid | attisdropped
-----+-----+-----+-----
tableoid  |      -6 | oid      | false
cmax      |      -5 | cid      | false
xmax      |      -4 | xid      | false
cmin      |      -3 | cid      | false
xmin      |      -2 | xid      | false
ctid      |      -1 | tid      | false
(6 rows)
```

Understanding System Columns

What is that 0 & 1??

The first number is OID and OID is always 0.

Since PostgreSQL 12, OIDs are no longer automatically assigned by default. The '0' indicates no OID was assigned for row.

Another digit 1 is for insertion of metadata event though there is no user define column.

```
[postgres=# select tableoid, xmin, cmin, xmax, cmax, ctid, * from nocol;
 tableoid |  xmin  |  cmin  |  xmax  |  cmax  |  ctid
-----+-----+-----+-----+-----+-----
    16517 | 311091 |      0 |      0 |      0 | (0,1)
(1 row)
```

Details added by the PostgreSQL system

So lets try one insert with real data

```
[postgres=# create database basics;
CREATE DATABASE
[postgres=# \c basics
You are now connected to database "basics" as user "postgres".
[basics=# create table test (id int, data char(10));
CREATE TABLE
[basics=# insert into test values (1,'first');
INSERT 0 1
basics=# █
```

What will be size of table?

Details added by the PostgreSQL system

Some background on our table

```
[basics=# \dt+ test
```

List of relations							
Schema	Name	Type	Owner	Persistence	Access method	Size	Description
public	test	table	postgres	permanent	heap	8192 bytes	

(1 row)

Things to notice: **Access method and Size**

```
[basics=# select * from test;
```

id	data
1	first

(1 row)

Remember: **SELECT * FROM test;**

* Means everything but without the system columns.

Details added by the PostgreSQL system

```
[basics=# select attrelid, attname, atttypid from pg_attribute
basics=# where attrelid = 'test'::regclass;
 attrelid | attname  | atttypid
-----+-----+-----
    18510 | tableoid |         26
    18510 | cmax     |         29
    18510 | xmax     |         28
    18510 | cmin     |         29
    18510 | xmin     |         28
    18510 | ctid     |         27
    18510 | id       |         23
    18510 | data     |       1042
(8 rows)
```



The two user defined columns we created

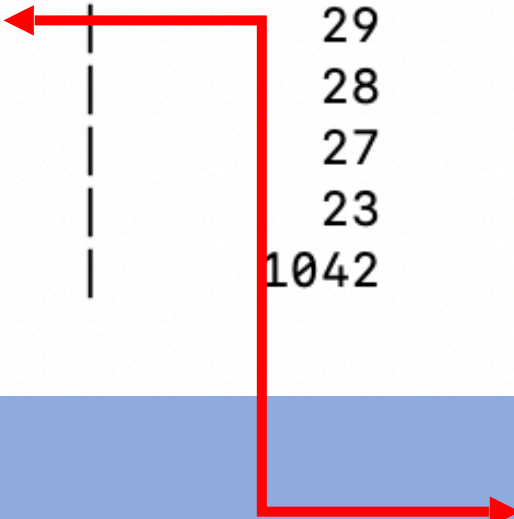
```
[postgres=# select oid,typename from pg_type catalog where typename='int4' or typename='bpchar';
 oid | typename
-----+-----
    23 | int4
   1042 | bpchar
(2 rows)
```


Details added by the PostgreSQL system

```
[basics=# select attrelid, attname, atttypid from pg_attribute  
basics=# where attrelid = 'test'::regclass;
```

attrelid	attname	atttypid
18510	tableoid	26
18510	cmax	29
18510	xmax	28
18510	cmin	29
18510	xmin	28
18510	ctid	27
18510	id	23
18510	data	1042

(8 rows)



Stuff added for us!

Details added by the PostgreSQL system

What the system added

Remember 18510 for later!!!

```
[basics=# select attrelid, attname, atttypid from pg_attribute  
basics=# where attrelid = 'test'::regclass;
```

attrelid	attname	atttypid
18510	tableoid	26
18510	cmax	29
18510	xmax	28
18510	cmin	29
18510	xmin	28
18510	ctid	27
18510	id	23
18510	data	1042

(8 rows)

Lets find out, where did the table data go with attrelid?

Where does data resides on file system

Where did that 18510 insert go?

```
[basics=# show data_directory;
          data_directory
-----
 /Library/PostgreSQL/17/data
(1 row)

[basics=# SELECT oid FROM pg_database WHERE datname = 'basics';
          oid
-----
    18509
(1 row)

[basics=# select pg_relation_filepath('test');
          pg_relation_filepath
-----
base/18509/18510
(1 row)
```

```
[root@c889f3eebc2d data # pwd
/Library/PostgreSQL/17/data
[root@c889f3eebc2d data # find . -name 18510
./base/18509/18510
[root@c889f3eebc2d data # ls -lhrt ./base/18509/18510
-rw-----@ 1 postgres  daemon  8.0K 20 Apr 09:15 ./base/18509/18510
```

What is in the data file

```
[root@c889f3eebc2d data #  
[root@c889f3eebc2d data # od -c ./base/18509/18510  
0000000  \0  \0  \0  \0 300  @ 273  8  \0  \0  \0  \0 034  \0 330 037  
0000020  \0      004      \0  \0  \0  \0  ?  **  N  \0  \0  \0  \0  \0  
0000040  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  
*  
0017720  \0  \0  \0  \0  \0  \0  \0  \0  \0  / 005  \0  \0  \0  \0  \0  
0017740  \0  \0  \0  \0  \0  \0  \0  \0  \0 001  \0 002  \0 002  \t 030  \0  
0017760 001  \0  \0  \0 027  f  i  r  s  t  \0  
0020000  
root@c889f3eebc2d data #
```

Not obvious where the 'id' went. But we can see where the 'data' is!

More data ...

Add a second row, start to see system columns

```
[basics=# insert into test values (22,'Second');
INSERT 0 1
[basics=# select xmin, xmax, * from test;
 xmin | xmax | id | data
-----+-----+----+-----
 1327 |    0 |  1 | first
 1328 |    0 | 22 | Second
(2 rows)
```

Before that let's quickly check the size again

```
[basics=# \dt+ test
              List of relations
 Schema | Name | Type  | Owner  | Persistence | Access method | Size  | Description
-----+-----+-----+-----+-----+-----+-----+-----
 public | test | table | postgres | permanent   | heap           | 8192 bytes |
(1 row)
```


Beyond the * data

```
[basics=# select tableoid, xmin, cmin, xmax, cmax, ctid,  
[basics=# * from test;
```

tableoid	xmin	cmin	xmax	cmax	ctid	id	data
18510	1327	0	0	0	(0,1)	1	first
18510	1328	0	0	0	(0,2)	22	Second

(2 rows)

Every table has several system columns that are implicitly defined by the system. These names cannot be used as names of user-defined columns.

```
[basics=# create table test1 ("xmax" int, id int);  
ERROR: column name "xmax" conflicts with a system column name
```

You do not really need to be concerned about these columns; just know they exist.

ctid

- GPS coordinate for every row in your table.
- (1,3) ->
 - The first number is the page number (block number in the table file)
 - The second number is the row's position within that page
 - So (1,3) means: page 1, row position 3
- CTIDs are not permanent! They can and will change when:
 - You run VACUUM FULL
 - The row gets updated
- Do NOT use CTIDs as long-term row identifiers.
- CTIDs are useful for:
 - Debugging
 - Understanding how PostgreSQL manages your data

ctid

```
[basics=# create table demo (id integer not null primary key, y int, z int);  
CREATE TABLE
```

```
[basics=# insert into demo values (1,2,3),(4,5,6);
```

```
INSERT 0 2
```

```
[basics=# select ctid, * from demo;
```

ctid	id	y	z
(0,1)	1	2	3
(0,2)	4	5	6

(2 rows)

```
[basics=# update demo set id = 6 where id = 1;
```

```
UPDATE 1
```

```
[basics=# select ctid, * from demo;
```

ctid	id	y	z
(0,2)	4	5	6
(0,3)	6	2	3

(2 rows)

```
[basics=# vacuum full demo;
```

```
VACUUM
```

```
[basics=# select ctid, * from demo;
```

ctid	id	y	z
(0,1)	4	5	6
(0,2)	6	2	3

(2 rows)

tableoid

- unique ID card for your table
- TableOID tells us which table, a row belongs to [unnecessary?]
- Becomes incredibly valuable
 - When working with inherited tables
 - When dealing with partitioned tables
 - When you need to join system catalogs to get table metadata”
- Use Case: For a large partitioned table {orders by year}:
 - Track which partition holds specific orders
 - Monitor partition usage
 - Troubleshoot data distribution issues

tableoid

```
[basics=# select relname from pg_class where oid=18510;
 relname
-----
 test
(1 row)
```

```
[basics=# create table part (id int, value int) partition by list (id);
CREATE TABLE
```

```
[basics=# create table p1 partition of part for values in (1);
CREATE TABLE
```

```
[basics=# create table p2 partition of part for values in (2);
CREATE TABLE
```

```
[basics=# insert into part values (1,100), (2,200);
INSERT 0 2
```

```
[basics=# select * from part;
```

id	value
1	100
2	200

(2 rows)

```
[basics=# select tableoid::regclass,* from part;
```

tableoid	id	value
p1	1	100
p2	2	200

(2 rows)

cmin and cmax

- Line numbers in your transaction's story.
- cmin : The command identifier (starting at zero) within the inserting transaction.
- cmax : The command identifier within the deleting transaction, or zero.
- RESET to 0 for each new transaction.
- Use them for debugging complex transactions

```

[postgres=# create table demo2(x int, y int);
CREATE TABLE
[postgres=# insert into demo2 values (1,1);
INSERT 0 1
[postgres=# -- This row gets cmin=0, cmax=0 (outside transaction)
[postgres=# begin;
BEGIN
[postgres=*# insert into demo2 values (2,2);
INSERT 0 1
[postgres=*# insert into demo2 values (3,3);
INSERT 0 1
[postgres=*# -- These are command 0 and command 1 (inside transaction)
[postgres=*# select cmin,cmax,* from demo2;
 cmin | cmax | x | y
-----+-----+---+---
      0 |      0 | 1 | 1
      0 |      0 | 2 | 2
      1 |      1 | 3 | 3
(3 rows)

```

```

[postgres=*# update demo2 set y=222 where x=2;
UPDATE 1
[postgres=*# -- this is command 2 inside transaction
[postgres=*# select cmin,cmax,* from demo2;
 cmin | cmax | x | y
-----+-----+---+---
      0 |      0 | 1 | 1
      1 |      1 | 3 | 3
      2 |      2 | 2 | 222
(3 rows)

```

xmin

- It's crucial for PostgreSQL's MVCC (Multi-Version Concurrency Control) system
- To identify different versions of the same row, PostgreSQL marks each of them with two values - XMIN, and XMAX to define the 'validity' of each row version.
- XMIN -> birth certificate for each row in your database
- The identity (transaction ID) of the inserting transaction for this row version.
- When you insert a row
 - The system assigns the current transaction ID as XMIN.
 - Other sessions querying the table will:
 - See this row only if their transaction started after yours committed
 - Not see it if their transaction started before yours committed [REPEATABLE READ]

xmax

- XMAX -> death certificate for a row version
- The identity (transaction ID) of the deleting transaction, or zero for an undeleted row version.
- XMAX can tell us several things:
 - XMAX = 0:
 - Row version is currently active
 - No transaction has marked it for update or delete
 - XMAX = transaction_id:
 - Row might be deleted OR updated
 - Transaction might be in progress
 - Transaction might have rolled back
 - Transaction might have completed but row remains visible

xmin and xmax

```
basics=# select xmin, xmax, * from demo;
```

xmin	xmax	id	y	z
1425	0	1	2	3
1425	0	4	5	6

(2 rows)

```
basics=# select pg_current_xact_id(), txid_current();  
pg_current_xact_id | txid_current
```

pg_current_xact_id	txid_current
1426	1426

(1 row)

```
[basics=# INSERT INTO demo VALUES (7,8,9);  
INSERT 0 1
```

```
[basics=# select xmin, xmax, * from demo;
```

xmin	xmax	id	y	z
1425	0	1	2	3
1425	0	4	5	6
1427	0	7	8	9

(3 rows)

Terminal #1

```
[basics=# select xmin, xmax, * from demo;
```

xmin	xmax	id	y	z
1425	0	1	2	3
1425	0	4	5	6
1427	0	7	8	9

(3 rows)

```
[basics=# start transaction;
```

```
START TRANSACTION
```

```
basics=# update demo set z=0 where id = 1 or id = 7;
```

```
UPDATE 2
```

```
basics=# select xmin, xmax, * from demo order by id;
```

xmin	xmax	id	y	z
1428	0	1	2	0
1425	0	4	5	6
1428	0	7	8	0

(3 rows)

Terminal #2 (not in the transaction)

```
[basics=# select xmin, xmax, * from demo;
```

xmin	xmax	id	y	z
1425	1428	1	2	3
1425	0	4	5	6
1427	1428	7	8	9

(3 rows)

$X_{max} > 0$ is telling us another version of the data is out there and the transaction id is 1428.

Also note the value of column z for $id = 1$ & $id = 7$

Terminal #3

```
basics=# update demo set z = 1 where id = 4;  
UPDATE 1  
basics=# █
```

Back to Terminal #1 ; still in transaction

```
[basics=# select xmin, xmax, * from demo order by id;
 xmin | xmax | id | y | z
-----+-----+----+---+---
 1428 |    0 |  1 | 2 | 0
 1429 |    0 |  4 | 5 | 1
 1428 |    0 |  7 | 8 | 0
(3 rows)
```

Because there was no lock on the rows in the transaction for Terminal number 3,
xmax = 0;
Xmin was incremented

Back to Terminal #2

```
[basics=# update demo set y=0;
```

HANGS!!!

- Gets blocked waiting for Transaction 1428 to commit

Terminal 1

```
basics=# select xmin, xmax, * from demo order by id;
```

xmin	xmax	id	y	z
1428	0	1	2	0
1429	0	4	5	1
1428	0	7	8	0

(3 rows)

```
basics=# commit;  
COMMIT
```

```
basics=# select xmin, xmax, * from demo order by id;
```

xmin	xmax	id	y	z
1430	1430	1	0	0
1430	0	4	0	1
1430	1430	7	0	0

(3 rows)

Terminal 2

```
basics=# update demo set y=0;  
UPDATE 3
```

```
basics=# select xmin, xmax, * from demo;
```

xmin	xmax	id	y	z
1430	1430	1	0	0
1430	1430	7	0	0
1430	0	4	0	1

(3 rows)

Terminal 3

```
basics=# select xmin, xmax, * from demo;
```

xmin	xmax	id	y	z
1430	1430	1	0	0
1430	1430	7	0	0
1430	0	4	0	1

(3 rows)

Thank you!!!