PERCONA

Databases run better with Percona

PERCONA

Do You Know A CID From An OID Or An XID? A beginners guide to the alphabet soup found in and around PostgreSQL tuples.

Dave Stokes

@Stoker

David.Stokes@Percona.com

https://speakerdeck.com/stoker

Who Am I

I am Dave Stokes

Technology Evangelist at Percona

Author of *MySQL & JSON - A Practical Programming Guide*

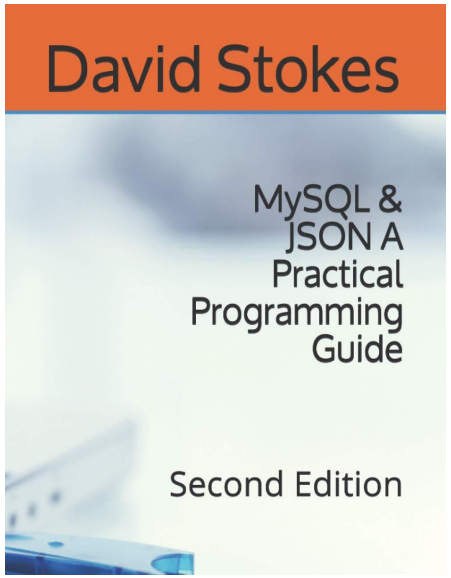
Over a decade on the Oracle MySQL Community Team

Started with MySQL 3.29

David.Stokes@Percona.com

@Stoker

https://speakerdeck.com/stoker



Do You Know A CID From An OID Or An XID? A beginners guide to the alphabet soup found in and around PostgreSQL tuples.

The learning curve for PostgreSQL can be nearly vertical for those caught up in the terminology.

So if you find yourself stuck between MIN and CMAX, or have no idea what those things are, then you should be in this talk.

This will be a gentle introduction for those new to PostgreSQL to some of the more common but no less obscure terminology that you will stumble over when trying to discern the manual pages or follow online discussions.

If you can not define a CID, XID, or OID, then you may want to attend to broaden your knowledge.

How Does PG Work?

The deep, dark secret nobody admits to!


HD
HISTORY.COM

XPS?

Acronyms can cause problems!

Starting with the basics

Ground rules

- The source code is the definitive answer
- followed by the documentation

Let us start with an insert

```
test=# CREATE TABLE example_1 (id int, a int);
CREATE TABLE
test=# INSERT INTO example_1 VALUES (1,2);
INSERT 0 1
```

What is that 0 & 1 stuff??

On successful completion, an INSERT command returns a command tag of the form

INSERT oid count

The count is the number of rows inserted or updated. oid is always 0

(it used to be the OID assigned to the inserted row if count was exactly one and the target table was declared WITH OIDS and 0 otherwise, but creating a table WITH OIDS is not supported anymore).

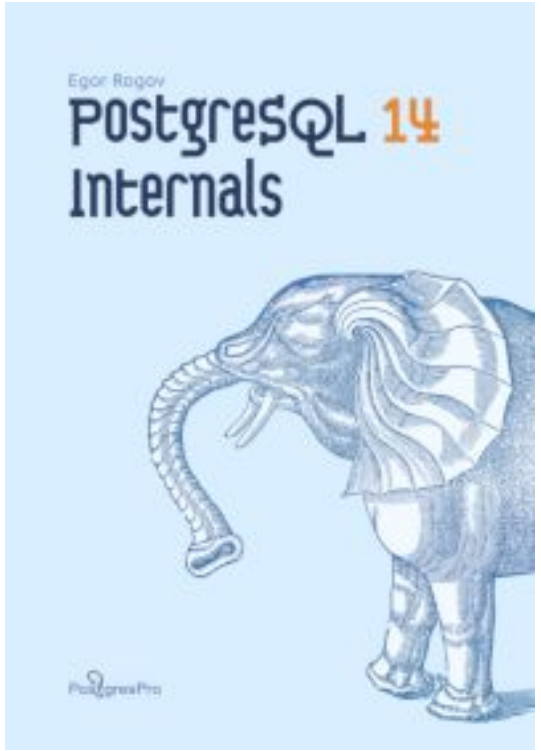
Not exactly intuitive

Thus you need two things handy at all times

Please keep these handy

1. PostgreSQL Manual
   a. https://www.postgresql.org/docs/16/index.html

2. PostgreSQL internals book
   a. https://edu.postgrespro.com/postgresql_internals-14_parts1-4_en.pdf



PERCONA

So lets try another insert

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

But where did that data go?

Some background on our table

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

basics=#
```

Things to notice:

- Access method
- Size

What the system added

basics=# select attrelid, attname, atttypid from pg_attribute where attrelid = 'a'::regclass;

attrelid	attname	atttypid
18103	tableoid	26
18103	cmax	29
18103	xmax	28
18103	cmin	29
18103	xmin	28
18103	ctid	27
18103	id	23
18103	data	1042

(8 rows)

Remember 18103 for later!!!

What the system added

```
basics=# select attrelid, attname, atttypid from pg_attribute
where attrelid = 'a'::regclass;
 attrelid | attname   | atttypid
----------+-----------+----------
    18103 | tableoid  |       26
    18103 | cmax      |       29
    18103 | xmax      |       28
    18103 | cmin      |       29
    18103 | xmin      |       28
    18103 | ctid      |       27
    18103 | id        |       23
    18103 | data      |     1042
(8 rows)
```

The two columns we created
with create table a (id int, data char(10));

What the system added

```
basics=# select attrelid, attname, atttypid from pg_attribute where attrelid = 'a'::regclass;
 attrelid | attname   | atttypid
----------+-----------+----------
    18103 | tableoid  |       26
    18103 | cmax      |       29
    18103 | xmax      |       28
    18103 | cmin      |       29
    18103 | xmin      |       28
    18103 | ctid      |       27
    18103 | id        |       23
    18103 | data      |     1042
(8 rows)
```

Stuff added for us!

Remember: SELECT * FROM foo;

* means everything but the system comums

Where did that 18103 insert go?

```
basics=# show data_directory;
     data_directory
----------------------------
 /var/lib/postgresql/16/main
(1 row)
```

```
root@test1:/var/lib/postgresql/16/main# find . -name 18103
./base/18102/18103
root@test1:/var/lib/postgresql/16/main# ls -lhrt ./base/18102/18103
-rw------- 1 postgres postgres 8.0K Mar 25 10:49 ./base/18102/18103
```

```
SELECT pg_relation_filepath('a');
```

```
basics=# show data_directory;
        data_directory
-------------------------------------
 C:/Program Files/PostgreSQL/15/data
(1 row)

basics=# select pg_relation_filepath('x');
 pg_relation_filepath
----------------------
 base/28051/28065
```

What is in that file

```
root@test1:/var/lib/postgresql/16/main# od -c !$
od -c ./base/18102/18103
0000000  \0  \0  \0  \0 270 337 264   ;  \0  \0  \0  \0 034  \0 330 037
0000020  \0      004      \0  \0  \0  \0 330 237   N  \0  \0  \0  \0  \0
0000040  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0017720  \0  \0  \0  \0  \0  \0  \0  \0 341 003  \0  \0  \0  \0  \0  \0
0017740  \0  \0  \0  \0  \0  \0  \0  \0 001  \0 002  \0 002  \b 030  \0
0017760 001  \0  \0  \0 027   f   i   r   s   t                   \0
0020000
```

root@test1:/var/lib/postgresql/16/main# od -c !$

```
od -c ./base/18102/18103
0000000  \0  \0  \0  \0 270 337 264   ;  \0  \0  \0  \0 034  \0 330 037
0000020  \0      004      \0  \0  \0  \0 330 237   N  \0  \0  \0  \0  \0
0000040  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0017720  \0  \0  \0  \0  \0  \0  \0  \0 341 003  \0  \0  \0  \0  \0  \0
0017740  \0  \0  \0  \0  \0  \0  \0  \0 001  \0 002  \0 002  \b 030  \0
0017760 001  \0  \0  \0 027   f   i   r   s   t                   \0
0020000
```

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

Alphabet Soup

More than the * data

```
basics=# select tableoid, xmin, cmin, xmax, cmax, ctid, * from a;
 tableoid | xmin | cmin | xmax | cmax | ctid  | id |    data
----------+------+------+------+------+-------+----+------------
    18103 |  993 |    0 |    0 |    0 | (0,1) |  1 | first
    18103 |  994 |    0 |    0 |    0 | (0,2) | 22 | Second
(2 rows)
```

Every table has several system columns that are implicitly defined by the system.

Therefore, these names cannot be used as names of user-defined columns. (Note that these restrictions are separate from whether the name is a keyword or not; quoting a name will not allow you to escape these restrictions.)

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

*** Unless you are the type to sit through a session on this stuff!**

ctid

The physical location (pointer) of the row version within its table.

Note that although the ctid can be used to locate the row version very quickly, a row's ctid will change if it is updated or moved by VACUUM FULL.

Therefore ctid is useless as a long-term row identifier.

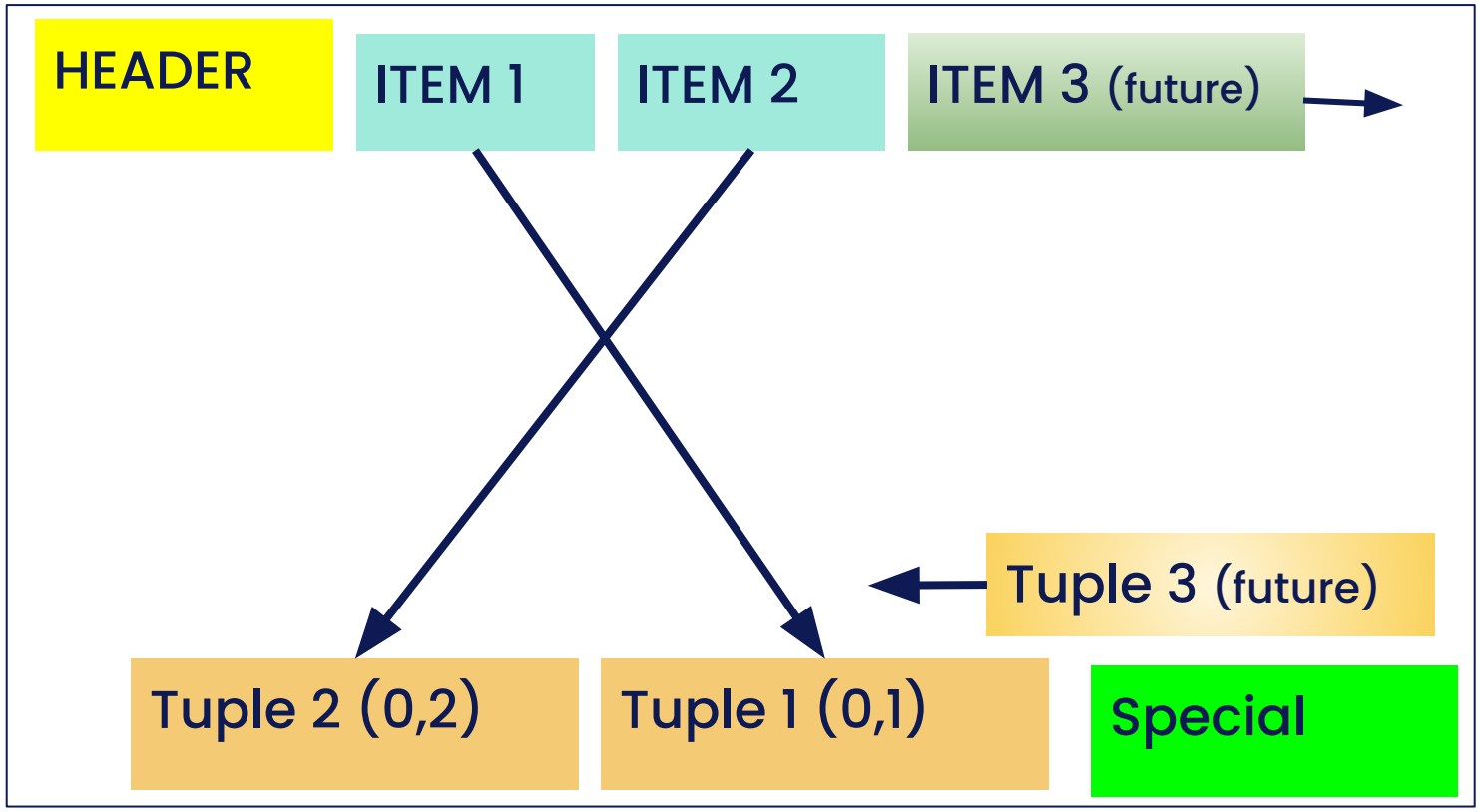
A primary key should be used to identify logical rows.

CTID

basics=# create table x (id integer not null primary key, y int, z int);
CREATE TABLE
basics=# insert into x values (1,2,3),(4,5,6);
INSERT 0 2
basics=# select ctid, * from x;

```
 ctid  | id | y | z
-------+----+---+---
 (0,1) |  1 | 2 | 3
 (0,2) |  4 | 5 | 6
(2 rows)
```

Page Layout Simplified – usually 8K is size



Add a second row, start to see system columns

```
basics=# insert into a values (22,'Second');
INSERT 0 1
basics=#
```

```
basics=# select xmin, xmax, * from a;
 xmin | xmax | id |     data
------+------+----+-------------
  993 |    0 |  1 | first
  994 |    0 | 22 | Second
(2 rows)
```

tableoid

The OID of the table containing this row.

This column is particularly handy for queries that select from partitioned tables or inheritance hierarchies , since without it, it's difficult to tell which individual table a row came from.

The tableoid can be joined against the oid column of pg_class to obtain the table name.

(select * from pg_class where oid=28055;)

Tableoid

```
basics=# select tableoid, * from x;
 tableoid | id | y | z
----------+----+---+---
    28058 |  1 | 2 | 3
    28058 |  4 | 5 | 6
    28058 |  7 | 8 | 9
(3 rows)
```

```
basics=# select relname
from pg_class
where oid=28058;
 relname
---------
 x
(1 row)
```

xmin

The identity (transaction ID) of the inserting transaction for this row version.

(A row version is an individual state of a row; each update of a row creates a new row version for the same logical row.)

When a row is created the xmin value is set to the Transaction of the INSERT statement.

xmax

The identity (transaction ID) of the deleting transaction, or zero for an undeleted row version.

It is possible for this column to be nonzero in a visible row version.

That usually indicates that the deleting transaction hasn't committed yet, or that an attempted deletion was rolled back.

To identify different versions of the same row, PG marks each of them with two values - XMIN, and XMAX to defineve the 'validity' of each row version.

When a row is DELETED the xmac of the current version is set to Transaction ID of the DELETE statement.

Consider (for now) that an UPDATE can be seen as two parts - a DELETE and an INSERT. The XMAX value of the current row is set to the transaction ID of the UPDATE. Then a new version of the tow with the XMIN set to the XMAX of the precious version.

xmin & xmax

```
basics=# select xmin, xmax, * from x;
 xmin | xmax | id | y | z
------+------+----+---+---
 1157 |    0 |  1 | 2 | 3
 1157 |    0 |  4 | 5 | 6
```

```
basics=# select pg_current_xact_id(), txid_current();
 pg_current_xact_id | txid_current
--------------------+--------------
               1158 |         1158
```

```
basics=# INSERT INTO x VALUES (7,8,9);
INSERT 0 1
basics=# select xmin, xmax, * from x where id=7;
 xmin | xmax | id | y | z
------+------+----+---+---
 1159 |    0 |  7 | 8 | 9
```

Terminal #1

```
basics=# start transaction;
START TRANSACTION
basics=*# update x set z=0 where id = 1 or id = 7;
UPDATE 2
basics=*# select xmin, xmax, * from x order by id;
 xmin | xmax | id | y | z
------+------+----+---+---
 1160 |    0 |  1 | 2 | 0
 1157 |    0 |  4 | 5 | 6
 1160 |    0 |  7 | 8 | 0
```

Terminal #2 (not in the transaction)

basics=# select xmin, xmax,* from x;

```
 xmin | xmax | id | y | z
------+------+----+---+---
 1157 | 1160 |  1 | 2 | 3
 1157 |    0 |  4 | 5 | 6
 1159 | 1160 |  7 | 8 | 9
```

Xmax > 0 is telling us another version of the data is out there and the transaction if is 1154.

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

Terminal #3

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

Back to Terminal #1 ; still in transaction

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

```
 xmin | xmax | id | y | z
------+------+----+---+---
 1160 | 1154 |  1 | 2 | 3
 1161 |    0 |  4 | 5 | 1   terminal 3
 1160 | 1154 |  7 | 8 | 9
```

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 x set y=0;

HANGS ! ! !

Terminals 1 & 2

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

```
 xmin | xmax | id | y | z
------+------+----+---+---
 1160 |    0 |  1 | 2 | 0
 1161 |    0 |  4 | 5 | 1
 1160 |    0 |  7 | 8 | 0
```

basics=*# commit;

```
COMMIT
```

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

```
 xmin | xmax | id | y | z
------+------+----+---+---
 1162 | 1162 |  1 | 0 | 0
 1162 |    0 |  4 | 0 | 1
 1162 | 1162 |  7 | 0 | 0
```

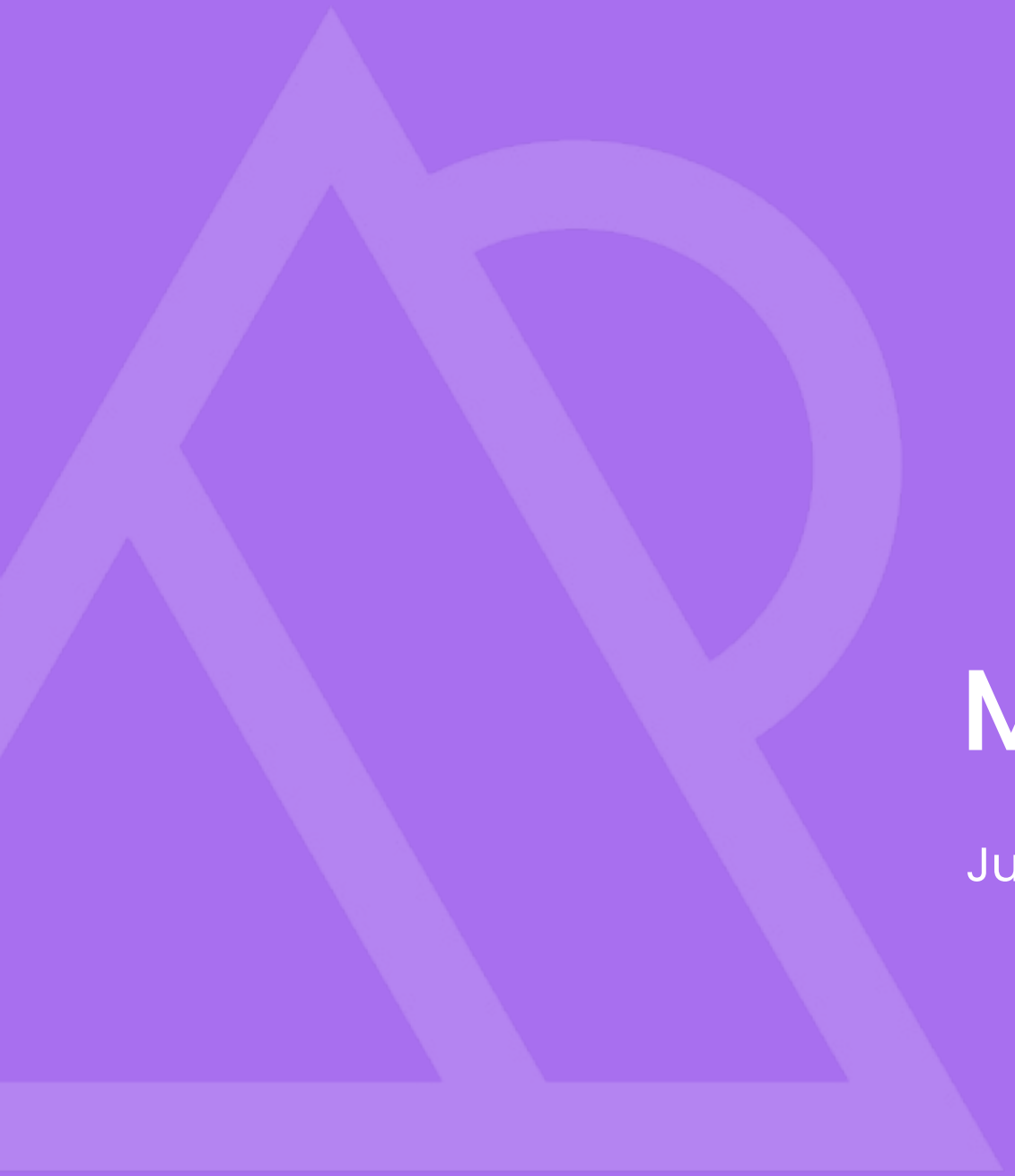
basics=# update x set y=0;

~~HANGS!!!~~

UPDATE 3

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

```
 xmin | xmax | id | y | z
------+------+----+---+---
 1162 | 1162 |  1 | 0 | 0
 1162 |    0 |  4 | 0 | 1
 1162 | 1162 |  7 | 0 | 0
```



More transaction stuff

Just what you wanted

cmin

The command identifier (starting at zero) within the inserting transaction.

cmax

The command identifier within the deleting transaction, or zero.

Those definitions were as clear as mud

‘*cmin*‘ and ‘*cmax*‘ are overlapped fields and are used within the same transaction to identify the command that changed a tuple.

Remember how xmin/xmax work for rows, cmin/cmax are the equivalent but inside a transaction!

```
basics=# insert into foo values (1,1);
INSERT 0 1
basics=# begin;
BEGIN
basics=*# insert into foo values (2,2);
INSERT 0 1
basics=*# select cmin,cmax, x, y from foo order by x;
 cmin | cmax | x | y
------+------+---+---
    0 |    0 | 1 | 1
    0 |    0 | 2 | 2
(2 rows)

basics=*# update foo set y=22 where x=2;
UPDATE 1
basics=*# select cmin,cmax, x, y from foo order by x;
 cmin | cmax | x | y
------+------+---+----
    0 |    0 | 1 |  1
    1 |    1 | 2 | 22
```

```
basics=*# insert into foo values (3,3);
INSERT 0 1
basics=*# select cmin,cmax, x, y from foo order by x;
 cmin | cmax | x | y
------+------+---+----
    0 |    0 | 1 |  1
    1 |    1 | 2 | 22
    2 |    2 | 3 |  3
(3 rows)

basics=*# update foo set y=222 where x=2;
UPDATE 1
basics=*# select cmin,cmax, x, y from foo order by x;
 cmin | cmax | x |  y
------+------+---+-----
    0 |    0 | 1 |   1
    3 |    3 | 2 | 222
    2 |    2 | 3 |   3
```

Whew!

Hopefully this filled in some knowledge gaps!

Percona is hiring!

- Senior Software Engineer (PostgreSQL)
- Support Engineer (PostgreSQL)
- PostgreSQL Evangelist

... and more!

THANK YOU!

David.Stokes@Percona.Com
@Stoker
speakerdeck.com/stoker

percona.com