Understanding MVCC, Vacuum, and how to monitor it with the Pg_Catalog

Boriss Mejias
Holistic System Software Engineer
Air Guitar Player

pgDay Paris 2023
March 23

EDB

Understanding MVCC, Vacuum, and how to monitor it with the Pg_Catalog

Boriss Mejias
Senior Solutions Architect
Air Guitar Player

pgDay Paris 2023
March 23

EDB

Prelude

ACID

Atomicity, Consistency, Isolation, Durability

Multi-Version Concurrency Control

Each row can have multiple versions
Only one version is visible to each observer
Time consistent view of the whole database
Snapshot

Everything runs in a transaction

```
INSERT INTO key_value (key, value)
VALUES (1905, 'Arsene Lupin');
```

Everything runs in a transaction

```
BEGIN;
INSERT INTO key_value (key, value)
VALUES (1905, 'Arsene Lupin');
COMMIT;
```

Concurrency

```
BEGIN;



SELECT value
  FROM key_value
 WHERE key = 1905;



COMMIT;
```

```
BEGIN;
UPDATE key_value
SET value = 'Arsène Lupin'
WHERE key = 1905;



COMMIT;
```

MVCC Basic Components

Each transaction has an identifier
`txid_current()`

Each row has visibility information
`xmin` and `xmax`

Each transaction has a status
`txid_status(txid)`

SQL write queries

`INSERT` sets **`xmin`**

`DELETE` sets **`xmax`**

`UPDATE` is delete and insert
sets **`xmax`**
creates a new version with **`xmin`**

SQL write queries

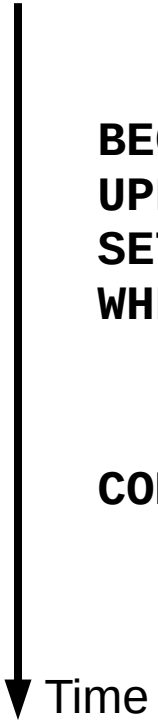
What about **TRUNCATE?**

Concurrency

```
BEGIN;        txid 1163


SELECT value
  FROM key_value
 WHERE key = 1905;


COMMIT;
```

txid 845

```
BEGIN;                txid 1190
UPDATE key_value
SET value = 'Arsène Lupin'
WHERE key = 1905;


COMMIT;
```



Multi-versions of the row

```
xmin xmax    key    value
 845 0       1905   'Arsene Lupin'
```

Multi-versions of the row

```
xmin xmax    key    value
 845 0       1905   'Arsene Lupin'

UPDATE →
xmin xmax    key    value
 845 1190    1905   'Arsene Lupin'
1190 0       1905   'Arsène Lupin'
```

Multi-versions of the row

```
xmin xmax    key    value
 845 0       1905   'Arsene Lupin'

UPDATE →
xmin xmax    key    value
 845 1190    1905   'Arsene Lupin'
1190 0       1905   'Arsène Lupin'
```

Commit state of txid 1190 will be “commited” or “aborted”

EDB POWER TO POSTGRES

Dead rows

DELETE creates a dead row

UPDATE: *delete* and insert → dead row

ROLLBACK of writes → dead row

Too many dead rows → bloated table

Interlude

Transaction Isolation Level

READ COMMITTED
REPEATABLE READ
SERIALIZABLE
READ UNCOMMITTED

Transaction Isolation Level

READ COMMITTED
REPEATABLE READ
SERIALIZABLE
~~READ UNCOMMITTED~~

Concurrency

```
BEGIN;
SELECT value
  FROM key_value
 WHERE key = 1905;
```

```
SELECT value
  FROM key_value
 WHERE key = 1905;

COMMIT;
```


```
BEGIN;
UPDATE key_value
SET value = 'Arsène Lupin'
WHERE key = 1905;

COMMIT;
```

Vacuum

VACUUM

VACUUM <table_name>;
Removes dead rows
Unless they are still potentially visible

VACUUM

```
BEGIN;
SELECT value
  FROM key_value
 WHERE key = 1905;
```

```
SELECT value
  FROM key_value
 WHERE key = 1905;
```


```
BEGIN;
UPDATE key_value
SET value = 'Arsène Lupin'
WHERE key = 1905;

COMMIT;
```

VACUUM

```
BEGIN;
SELECT value
  FROM key_value
 WHERE key = 1905;
```

```
SELECT value
  FROM key_value
 WHERE key = 1905;
```

```
BEGIN;
UPDATE key_value
SET value = 'Arsène Lupin'
WHERE key = 1905;

COMMIT;

VACUUM;
```

Time

Table pg_catalog.pg_stat_all_tables

```
-[ RECORD 1 ]-------+-----------
relid               | 24652
schemaname          | paris
relname             | key_value
n_tup_ins           | 11284004
n_tup_upd           | 6712
n_tup_del           | 425152
n_tup_hot_upd       | 713
n_live_tup          | 10858852
n_dead_tup          | 5666
n_mod_since_analyze | 6210
n_ins_since_vacuum  | 4851
```

VACUUM paris.key_value;

```
-[ RECORD 1 ]-------+-----------
relid               | 24652
schemaname          | paris
relname             | key_value
n_tup_ins           | 11284004
n_tup_upd           | 6712
n_tup_del           | 425152
n_tup_hot_upd       | 713
n_live_tup          | 10858852
n_dead_tup          | 0
n_mod_since_analyze | 6210
n_ins_since_vacuum  | 0
```

VACUUM ANALYZE paris.key_value;

```
-[ RECORD 1 ]-------+-----------
relid               | 24652
schemaname          | paris
relname             | key_value
n_tup_ins           | 11284004
n_tup_upd           | 6712
n_tup_del           | 425152
n_tup_hot_upd       | 713
n_live_tup          | 10858852
n_dead_tup          | 0
n_mod_since_analyze | 0
n_ins_since_vacuum  | 0
```

VACUUM

VACUUM <table_name>;
Removes dead rows
Unless they are still potentially visible

VACUUM

VACUUM <table_name>;
Removes dead rows
Unless they are still potentially visible

Hunt ‘idle in transaction’ sessions!!
And long running queries!!

Use pg_catalog.pg_stat_activity

```
SELECT pid
FROM pg_stat_activity
WHERE state = 'idle in transaction';

SELECT pid
     , query
     , wait_event_type
     , wait_event
     , now() - query_start
FROM pg_stat_activity
WHERE state='active'
AND now()-query_start > interval '15 minutes';
```

Use pg_catalog.pg_stat_activity

```
SELECT pg_terminate_backend(pid)
FROM pg_stat_activity
WHERE state = 'idle in transaction';

SELECT pg_cancel_backend(pid)
     , query
     , wait_event_type
     , wait_event
     , now() - query_start
FROM pg_stat_activity
WHERE state='active'
AND now()-query_start > interval '15 minutes';
```

VACUUM effects

VACUUM locks against DDL
INSERT, DELETE, UPDATE can still run
It does not reduce size of tables

VACUUM effects

VACUUM locks against DDL
INSERT, DELETE, UPDATE can still run
It does not reduce size of tables

Divertimento or delude

VACUUM FULL
Locks Everything
It creates a new table
Think of a butterfly

autovacuum

Autovacuum

It runs **VACUUM** and **ANALYZE**
Runs all the time
No scheduling, just nap times
It cancels itself to avoid blocking user's actions

Table pg_catalog.pg_stat_all_tables

```
-[ RECORD 1 ]-------+-----------
relid               | 24652
schemaname          | paris
relname             | key_value
last_vacuum         | 2023-02-19 12:58:42
last_autovacuum     | 2023-03-22 07:06:06
last_analyze        | 2023-02-19 12:58:42
last_autoanalyze    | 2023-03-23 06:23:16
vacuum_count        | 7
autovacuum_count    | 6128
analyze_count       | 10676
autoanalyze_count   | 7
```

Check the progress

Table pg_catalog.pg_stat_progress_vacuum

```
-[ RECORD 1 ]------+-----------
pid                | 27666
datid              | 18613
datname            | paris
relid              | 24652
phase              | performing final cleanup
heap_blks_total    | 1
heap_blks_scanned  | 1
heap_blks_vacuumed | 1
index_vacuum_count | 0
max_dead_tuples    | 291
num_dead_tuples    | 0
```

Table pg_catalog.pg_stat_progress_analyze

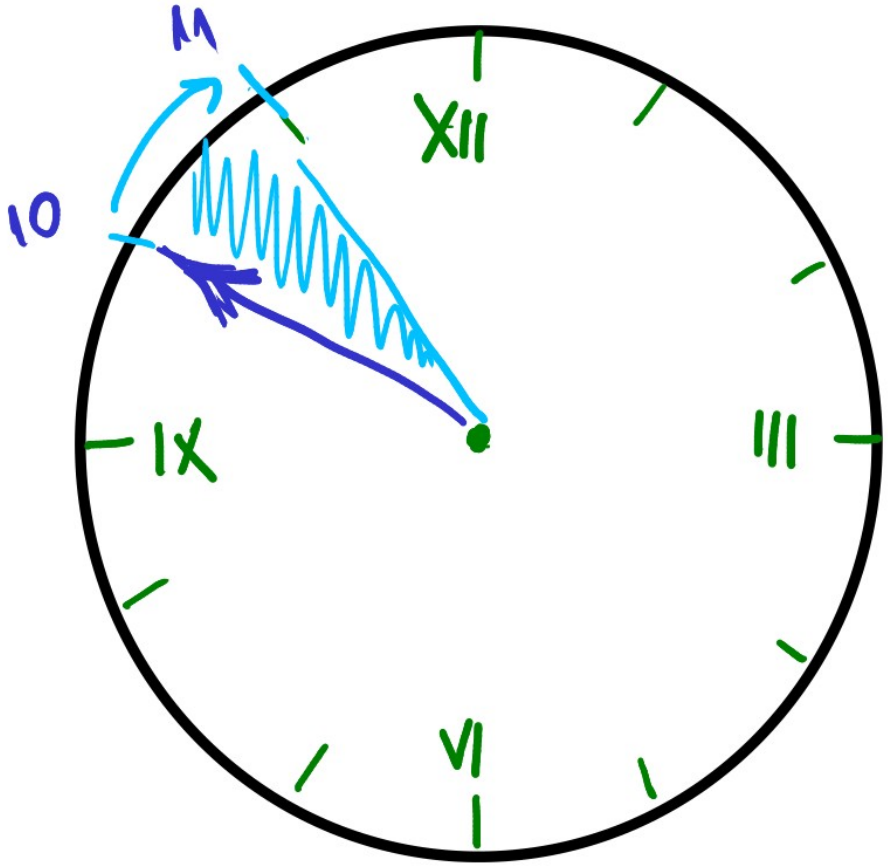
```
-[ RECORD 1 ]-------------+-----------
pid                       | 3012146
datid                     | 18613
datname                   | paris
relid                     | 66209
phase                     | acquiring sample rows
sample_blks_total         | 30000
sample_blks_scanned       | 3454
ext_stats_total           | 0
ext_stats_computed        | 0
child_tables_totals       | 0
child_tables_done         | 0
current_child_table_relid | 0
```

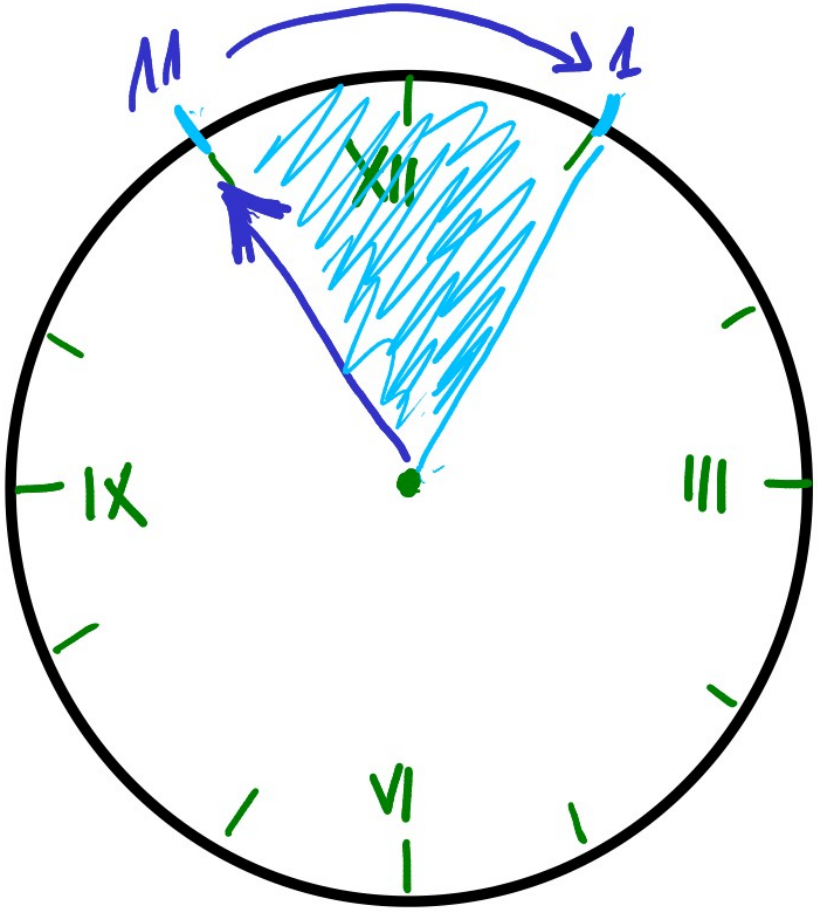
At the pub

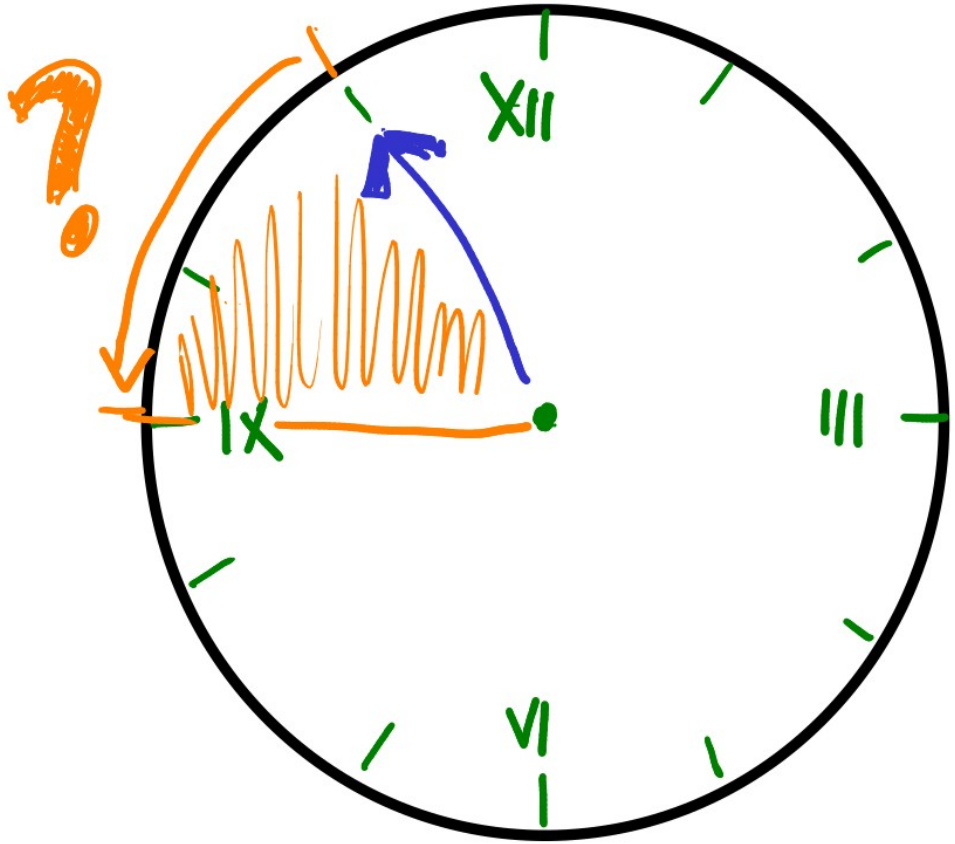
Freezing

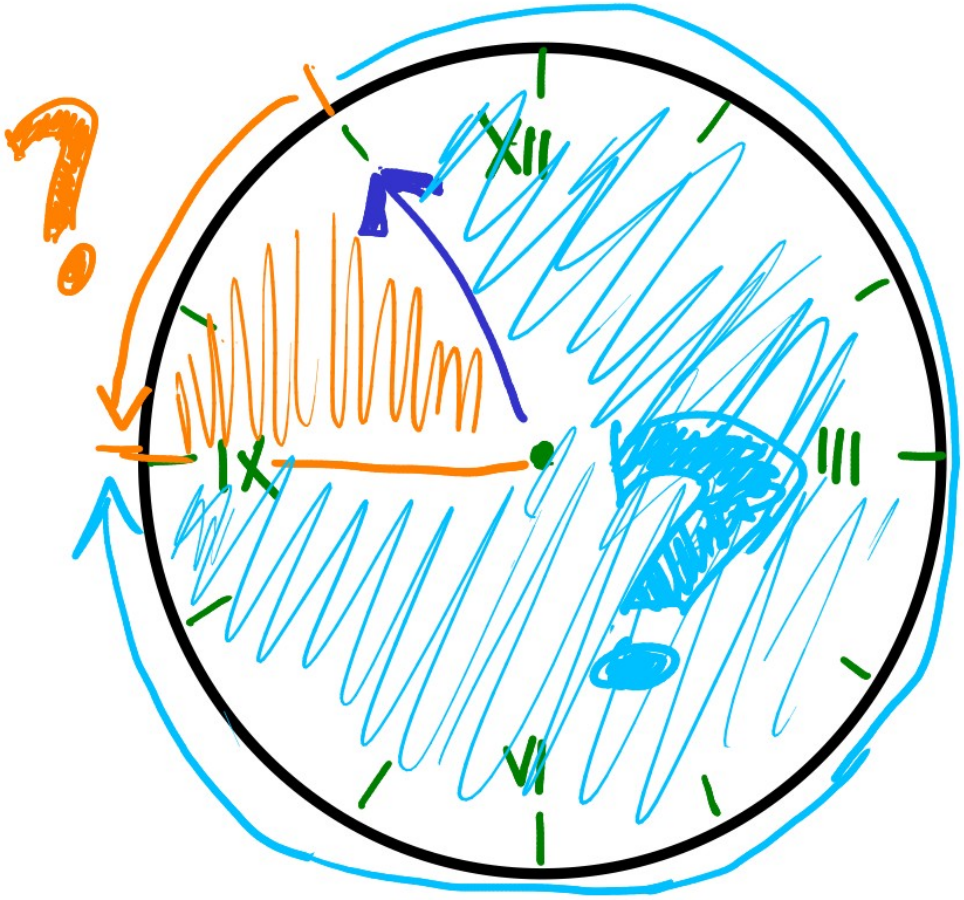
Transactions are 4 byte integers

Counter wraps every 4 billion xids


11
10
XII
IX
III
VI


11
1
XII
IX
III
VI


?
XII
III
VI
IX


?
XII
III
VI
IX

Freezing

Transactions are 4 byte integers

Counter wraps every 4 billion xids

Visibility of 2 billion xids to the past

Very old rows get frozen

Table pg_catalog.pg_database

```
datname        | name
datdba         | oid
encoding       | integer
datcollate     | name
datctype       | name
datistemplate  | boolean
datallowconn   | boolean
datconnlimit   | integer
datlastsysoid  | oid
datfrozenxid   | xid
datminmxid     | xid
dattablespace  | oid
datacl         | aclitem[]
```

Table pg_catalog.pg_database

```
datname        | name
datdba         | oid
encoding       | integer
datcollate     | name
datctype       | name
datistemplate  | boolean
datallowconn   | boolean
datconnlimit   | integer
datlastsysoid  | oid
datfrozenxid   | xid
datminmxid     | xid
dattablespace  | oid
datacl         | aclitem[]
```

EDB
POWER TO POSTGRES

Table pg_catalog.pg_database

```
SELECT datname
     , datfrozenxid
     , age(datfrozenxid)
FROM pg_database;
```

Finale

MVCC and VACUUM

MVCC crucial for ACID

Dead rows are created → Maintenance

VACUUM removes dead rows

Autovacuum does it for you

Plenty of data exposed at the catalog

Thank you!