EDB™ PgDay CWB 2023

Entendendo Transaction Wraparound

Israel Barth Rubio - Software Engineer
21 de Outubro de 2023

MVCC

- Multi-version concurrency control
- Cria novas versões de tuplas e retém as anteriores
- Sem bloqueio de **leitura X escrita**
- Visibilidade baseada em commit log, transaction ID, isolation level

EDB

Transaction ID

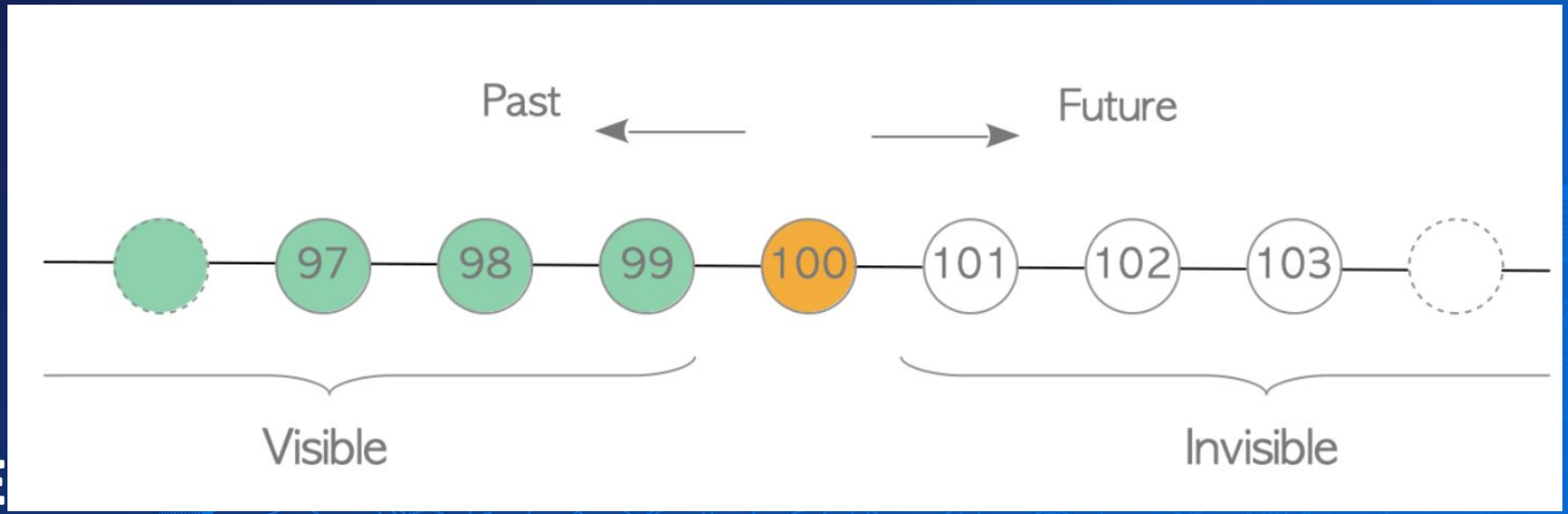
- Conhecido como **txid**
- Representado por um inteiro sem sinal de 32 bits

```
testdb=# BEGIN;
BEGIN
testdb=# SELECT txid_current();
 txid_current
--------------
          100
(1 row)
```

EDB

Transaction ID

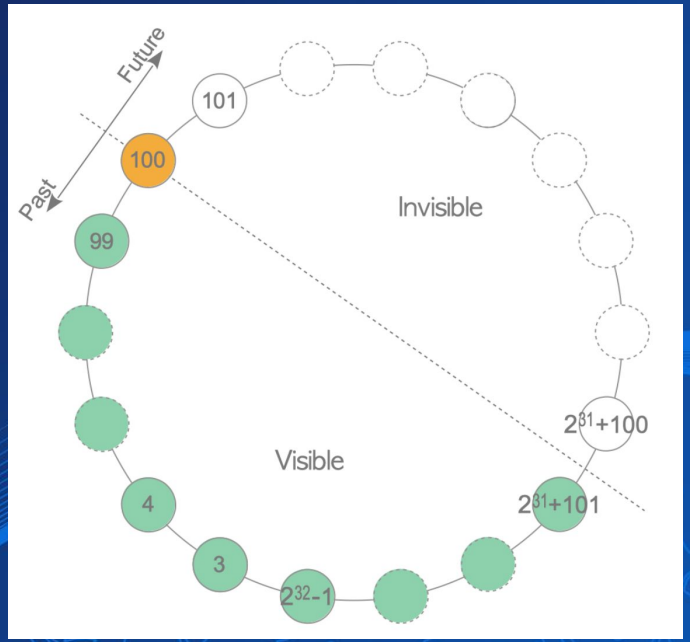
- Utilizado para checagem de visibilidade



Transaction ID

- O que acontece ao chegar no ID $2^{32} - 1$?
  - O próximo **txid** vai reiniciar do **0**
- E o que acontece com as checagens de visibilidade?
  - São feitas considerando um espaço circular

Transaction ID



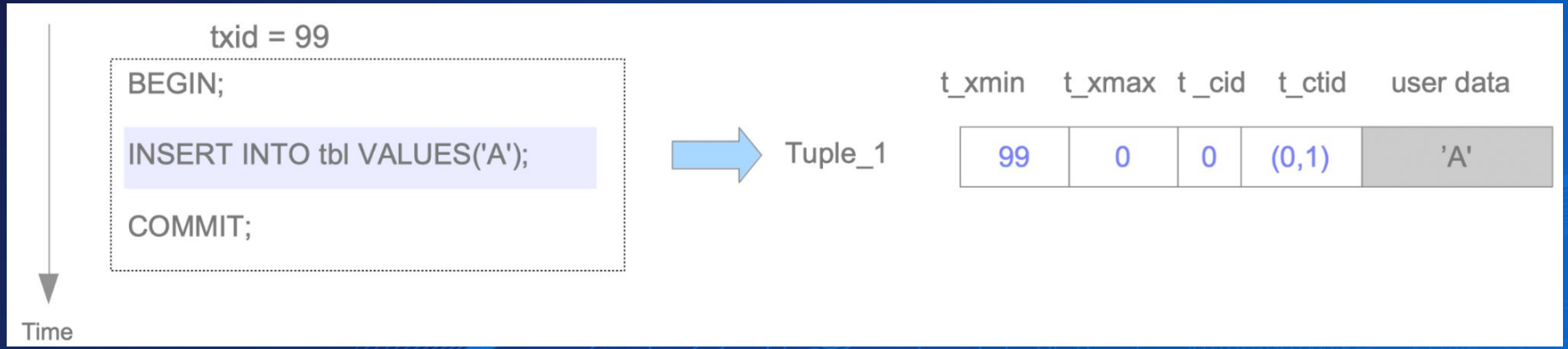
Formato das tuplas

Formato das tuplas

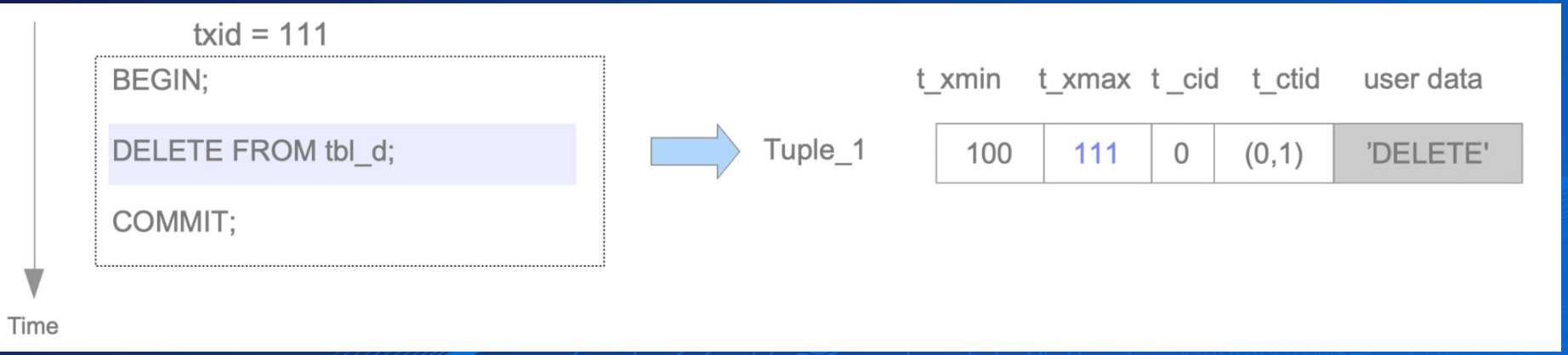
- **t_xmin**: **txid** que criou a versão da tupla
- **t_xmax**: **txid** que removeu a versão da tupla, se for o caso. **0** caso contrário
- **t_cid**: sequência do comando dentro da transação que criou a versão da tupla, começando com **0**
- **t_ctid**: ponteiro para a própria versão da tupla. Quando um comando **UPDATE** é executado uma nova versão da tupla é criada, e então o **t_ctid** vira um ponteiro para a nova versão

EDB

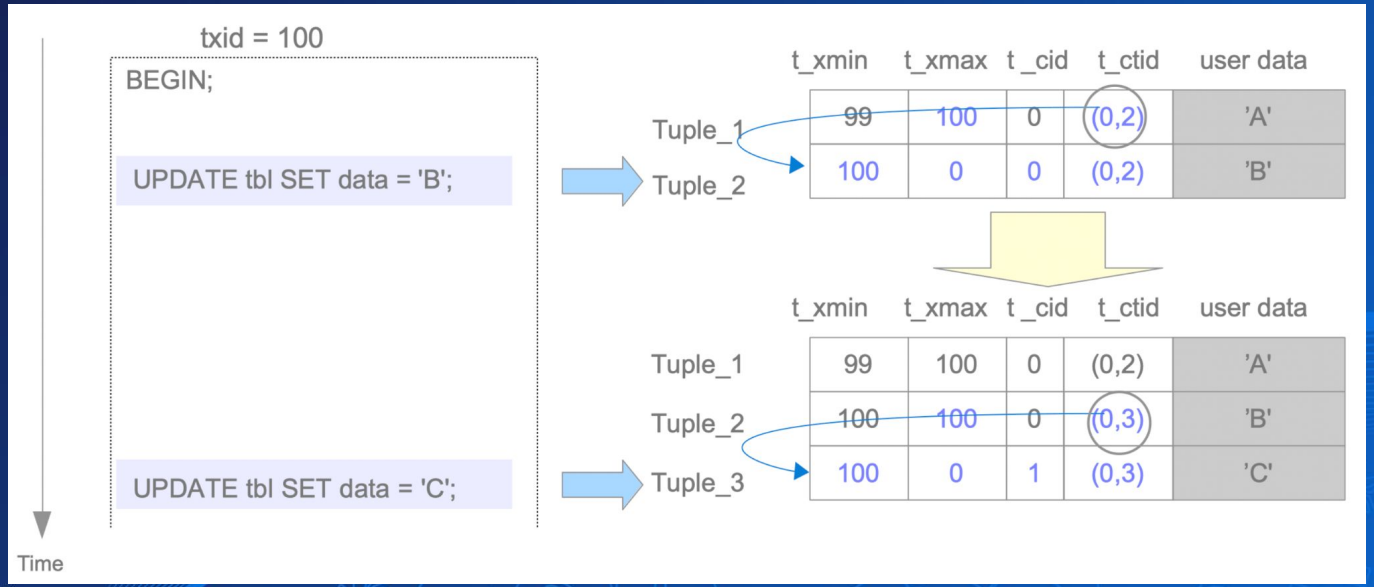
INSERT



DELETE



UPDATE

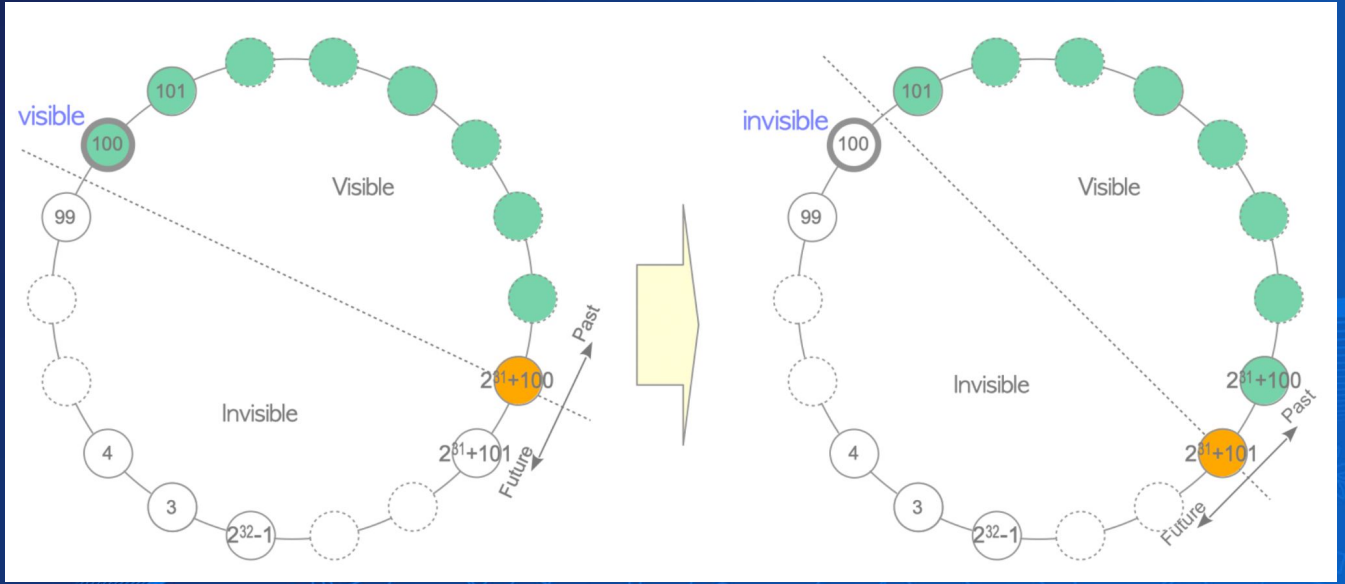


Visibilidade das tuplas

- **dead tuple**: versão de tupla que não é mais visível para nenhuma transação
- **live tuple**: versão de tupla que ainda é visível para uma transação em andamento, ou que será visível para novas transações

EDB

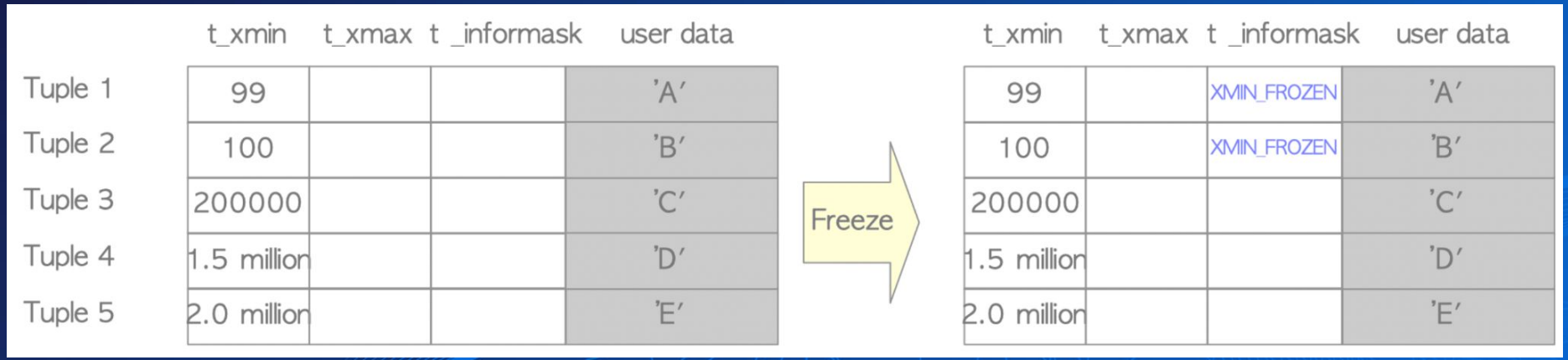
Transaction Wraparound



Freezing

- Tarefa feita pelo **VACUUM**
- Marca **live tuples** muito antigas como “congeladas”
  - Elas são tão antigas que é garantido que elas devem ser visíveis para qualquer transação em andamento ou futura!
- **xmin horizon**: limita até qual **txid** pode ser congelada
  - Não se deve congelar uma tupla que não pode ser visível para todas as transações!

Freezing



Freezing

- Funciona no conceito de “idade”:
  - **Idade da tupla**: diferença entre a **txid** mais atual no cluster e o **xmin** da tupla
  - **Idade da tabela**: idade da tupla mais velha da tabela
  - **Idade do banco de dados**: idade da tabela mais velha da banco de dados
  - **Idade da instância**: idade do banco de dados mais velho da instância

EDB

Freezing

- **vacuum_freeze_min_age**:
  - Valor padrão: **50M**
  - Checa tuplas que estão em páginas sendo lidas pelo **VACUUM**
  - Congela tuplas que satisfazem a fórmula:
    - **t_xmin < OldestXmin - vacuum_freeze_min_age**

Freezing

- **vacuum_freeze_table_age**:
  - Valor padrão: **150M**
  - Lê todas as páginas da tabela que tem 1 ou mais tuplas não congeladas quando a idade da tabela chegar nesse limite

Freezing

- **autovacuum_freeze_max_age**:
  - Valor padrão: **200M**
  - Similar ao anterior, mas mais agressivo:
    - **VACUUM** roda mesmo que **autovacuum** esteja desabilitado
    - Não é cancelável
  - Log mensagens como essa: **LOG: autovacuum XXX.YYY (to prevent wraparound)**

Condições limitantes

- Qualquer coisa que impeça o **xmin horizon** de avançar na instância:
  - Transações rodando por muito tempo
  - Transações rodando por muito tempo em um standby que tem **hot_standby_feedback** habilitado
  - Prepared transactions
  - Replication slots

EDB

Condições limitantes

- Se as condições limitantes persistirem por tempo suficiente para que o PostgreSQL esteja próximo de não ter mais como consumir **txid**, esses tipos de mensagens podem aparecer nos logs:
  - **WARNING: oldest xmin is far in the past**
  - **WARNING: database "XXX" must be vacuumed within YYY transactions**
  - **WARNING: database with OID XXX must be vacuumed within YYY transactions**

EDB

Condições limitantes

- Se as condições limitantes persistirem por tempo suficiente para que o PostgreSQL não seja mais capaz de consumir **txid**, esses tipos de mensagens podem aparecer nos logs:
  - **ERROR: database is not accepting commands to avoid wraparound data loss in database "XXX"**
  - **ERROR: database is not accepting commands to avoid wraparound data loss in database with OID XXX**

EDB

Lidando com o problema

- Você pode enfrentar duas situações:
  - Instância está prestes a desligar em emergência:
    - Aqui a instância ainda está operável e você pode seguir com os passos
  - Instância já foi desligada em emergência:
    - Nesse caso a instância não está mais operável, e você precisa iniciar ela em modo de usuário único:
      - **postgres --single -D <data_directory> <database_name>**

Lidando com o problema

- Em ambos os casos você precisa:
  - Eliminar as condições limitantes
  - Rodar **VACUUM** nos objetos antigos

EDB

Lidando com o problema

- Identificar as condições limitantes:

```
SELECT (SELECT coalesce(max(pg catalog.age(backend_xmin)), 0)
        FROM pg catalog.pg stat activity
        WHERE state != 'idle') AS oldest xact age,
       (SELECT coalesce(max(pg_catalog.age(backend_xmin)), 0)
        FROM pg catalog.pg stat replication) AS oldest standby_xact_age,
       (SELECT coalesce(max(pg catalog.age(transaction)), 0)
        FROM pg catalog.pg prepared xacts) AS oldest prepared_xact_age,
       (SELECT coalesce(max(pg catalog.age(xmin)), 0)
        FROM pg_catalog.pg_replication_slots) AS oldest_replication_slot_age;
```

Lidando com o problema

- Identificar os clientes causando o problema, se for o caso:

```
SELECT pid,
       age(backend_xmin)
       datname,
       usename,
       application_name,
       client_addr,
       state,
       backend_type,
       query
FROM pg_catalog.pg_stat_activity
WHERE state != 'idle'
  AND pg_catalog.age(backend_xmin) >
pg_catalog.current_setting('autovacuum_freeze_max_age')::bigint
ORDER BY 2 DESC
LIMIT 10;
```

Lidando com o problema

- Eliminar esses clientes, se for o caso:

```
-- Com base nos PIDs retornados pela query anterior
SELECT pg_catalog.pg_cancel_backend(pid);
-- ou
SELECT pg_catalog.pg_terminate_backend(pid);
```

Lidando com o problema

- Identificar os standbys causando o problema, se for o caso:

```
SELECT pid,
       pg_catalog.age(backend_xmin),
       usename,
       application_name,
       client_addr,
       state
FROM pg_catalog.pg_stat_replication
WHERE pg_catalog.age(backend_xmin) >
pg_catalog.current_setting('autovacuum_freeze_max_age')::bigint
ORDER BY 2 DESC
LIMIT 10;
```

Lidando com o problema

- Eliminar os clientes causando o problema nesses standbys, se for o caso:

```
-- Você precisa identificar os PIDs nos standbys
-- O PID nesse caso não é o que foi retornado pela query anterior
SELECT pg_catalog.pg_cancel_backend(pid);
-- ou
SELECT pg_catalog.pg_terminate_backend(pid);
```

Lidando com o problema

- Identificar as prepared transactions causando o problema, se for o caso:

```
SELECT transaction,
       gid,
       pg_catalog.age(transaction),
       prepared,
       owner,
       database
FROM pg prepared xacts
WHERE pg_catalog.age(transaction) >
pg catalog.current_setting('autovacuum_freeze_max_age')::bigint
ORDER BY 3 DESC
LIMIT 10;
```

Lidando com o problema

- Encerrar essas prepared transactions, se for o caso:

```
-- Use o gid retornado pela query anterior
COMMIT PREPARED 'gid';
-- ou
ROLLBACK PREPARED 'gid';
```

Lidando com o problema

- Identificar os replication slots causando o problema, se for o caso:

```
SELECT slot_name,
       pg_catalog.age(xmin),
       slot_type,
       database,
       temporary,
       active,
       active_pid
FROM pg_catalog.pg_replication_slots
WHERE pg_catalog.age(xmin) >
pg_catalog.current_setting('autovacuum_freeze_max_age')::bigint
ORDER BY 2 DESC
LIMIT 10;
```

Lidando com o problema

- Destruir esses replication slots, se for o caso:

```
-- Utilize o slot_name retornado pela query anterior
SELECT pg_catalog.pg_drop_replication_slot(slot_name := 'slot_name');
```

Lidando com o problema

- Identifique os bancos de dados mais antigos na instância:

```
SELECT datname,
       pg_catalog.age(datfrozenxid)
FROM pg_catalog.pg_database
WHERE pg_catalog.age(datfrozenxid) >
pg_catalog.current_setting('autovacuum_freeze_max_age')::bigint
ORDER BY 2 DESC
LIMIT 10;
```

Lidando com o problema

- Em cada um dos bancos antigos você precisa:
  - Identificar os objetos antigos nele
  - Rodar **VACUUM** nesses objetos
    - Você pode iniciar um **VACUUM** global no banco de dados:
      - `VACUUM;`
    - Ou você pode identificar os objetos mais antigos com essa query:

EDB

Lidando com o problema

```
SELECT c.oid::regclass AS table name,
       greatest(pg_catalog.age(c.relfrozenxid), pg_catalog.age(t.relfrozenxid)) AS
age
FROM pg catalog.pg class c
LEFT JOIN pg catalog.pg class t
       ON c.reltoastrelid = t.oid
WHERE c.relkind IN ('r', 'm')
  AND greatest(pg_catalog.age(c.relfrozenxid), pg_catalog.age(t.relfrozenxid)) >
pg catalog.current_setting('autovacuum_freeze_max_age')::bigint
ORDER BY 2 ASC
LIMIT 10;
```

Lidando com o problema

- De qualquer forma, **não** use **VACUUM FREEZE**!
  - A opção **FREEZE** significa rodar **VACUUM** com:
    - **vacuum_freeze_min_age = 0**
    - **vacuum_freeze_table_age = 0**
  - E isso só vai atrasar toda a operação

Monitoramento

- **txid** restantes até que o PostgreSQL seja desligado em emergência:

```
SELECT 2^31 - max(x.age) AS txid_left
FROM (
   SELECT coalesce(max(pg_catalog.age(backend_xmin)),0) AS age
   FROM pg_catalog.pg_stat_activity
   WHERE state != 'idle'
   UNION ALL
   SELECT coalesce(max(pg_catalog.age(backend_xmin)),0) AS age
   FROM pg_catalog.pg_stat_replication
   UNION ALL
   SELECT coalesce(max(pg_catalog.age(transaction)), 0) AS age
   FROM pg_catalog.pg_prepared_xacts
   UNION ALL
   SELECT coalesce(max(pg_catalog.age(xmin)), 0) AS age
   FROM pg_catalog.pg_replication_slots
) x
```

Monitoramento

- Objetos que devem enfrentar um **VACUUM** mais agressivo em breve:

```
SELECT c.oid::regclass AS table name,
       pg catalog.current setting('autovacuum freeze max age')::bigint -
greatest(pg catalog.age(c.relfrozenxid), pg_catalog.age(t.relfrozenxid)) AS
xid left to antifreeze
FROM pg_catalog.pg_class c
LEFT JOIN pg catalog.pg class t
       ON c.reltoastrelid = t.oid
WHERE c.relkind IN ('r','m')
ORDER BY 2 ASC
LIMIT 10;
```

Recomendações

- Implemente monitoramento das suas instâncias PostgreSQL
- Torne as transações tão curtas quanto possível

Leitura complementar

- Detalhes internos do PostgreSQL: https://www.interdb.jp/pg/pgsql05.html
  - Contém informações mais detalhadas sobre alguns tópicos que vimos hoje
  - Todos as imagens de exemplo desta apresentação foram retiradas desse site
- Estude sobre **multixactid** wraparound
  - O mecanismo é relativamente parecido com o que foi apresentado hoje, envolvendo **multixactid**

EDB

Perguntas?

- Estou à disposição também por meio do:
  - Postgres workspace no Slack: Israel Barth
  - Email: barthisrael@gmail.com