Detecting and fixing corruptions

adyen

engineered
for ambition

Introduction story:

I am afraid of three things.
Messing up production data
Corruptions
My wife when she is angry with me :D

This deck contains multiple ways to corrupt your database

Be very careful to use this on a live database

and don't blame me when you screw up

Derk van Veen

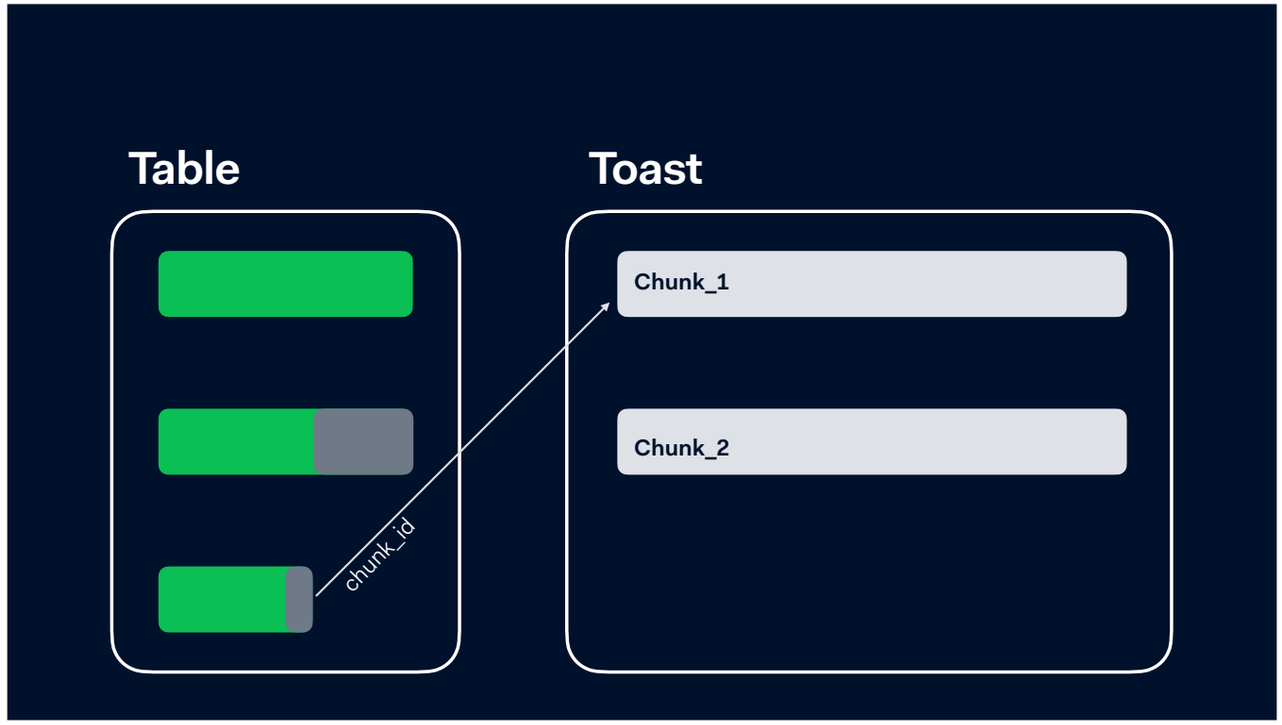
Corruption doctor

Adyen

adyen

engineered
for ambition

The next best thing after sliced bread.
The Oversized-Attribute Storage Technique, used to store text, json, bytea
Automatically triggered for rows exceeding 2kb

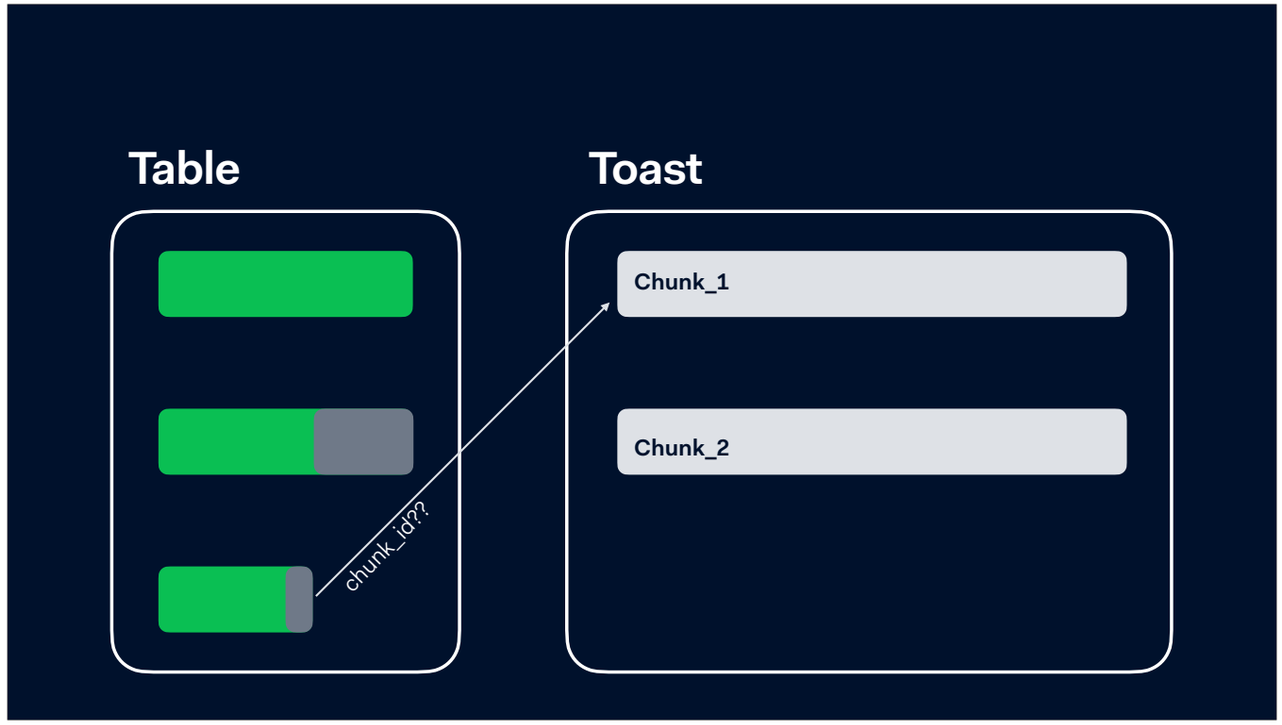


Uitleg over concepten toast.
Een chunk kan uit meerdere sequences bestaan.

```
create table test_toast(spread varchar);
insert into test_toast values (’little spread');
insert into test_toast (SELECT '800 length spread:  ' | <800 - 19  random chars> );
insert into test_toast (SELECT '2500 length spread: ' | <2500 - 20 random chars> );
insert into test_toast (SELECT '5000 length spread: ' | <5000 - 20 random chars> );
```

```
select ctid, octet_length(spread) from test_toast;
 ctid  | octet_length
-------+--------------
 (0,1) |           12
 (0,2) |          800
 (0,3) |         2500
 (0,4) |         5000
```

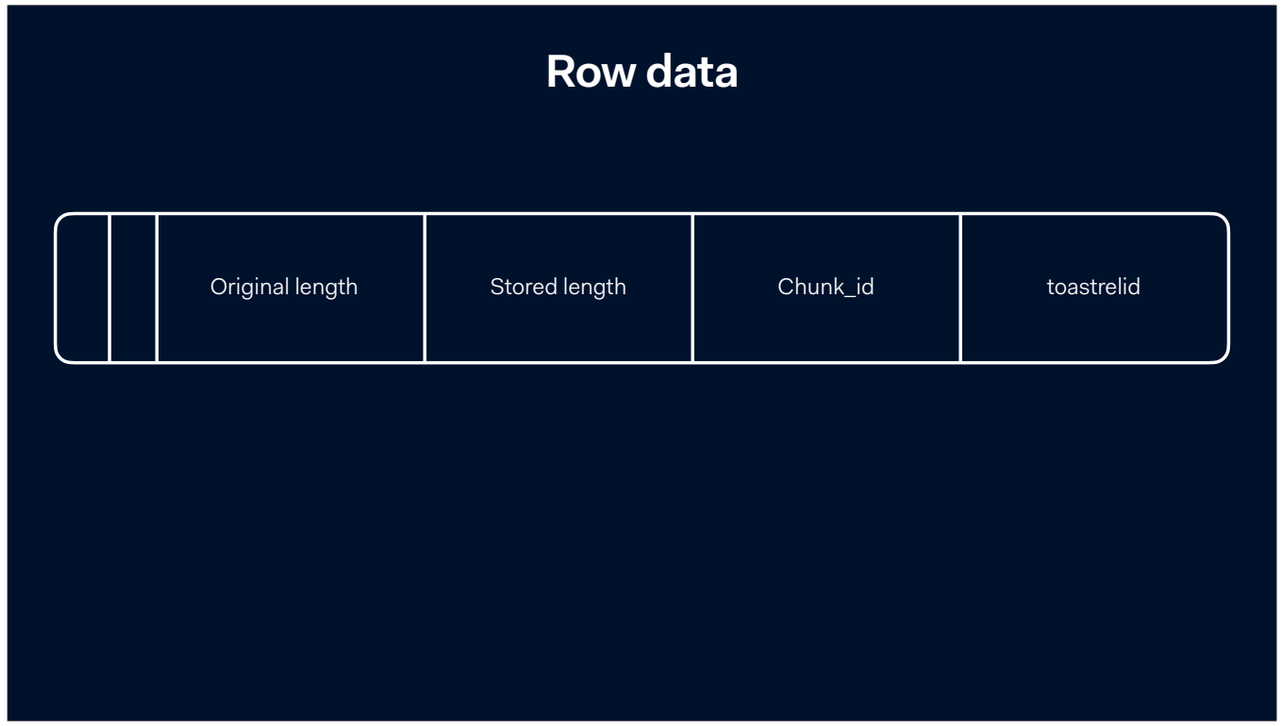
octet_length is the number of bytes.



Uitleg over concepten toast.
Een chunk kan uit meerdere sequences bestaan.

```
SELECT ctid, xmin, xmax from test_toast ;
 ctid  |   xmin   | xmax
-------+----------+------
 (0,1) | 60842144 |    0
 (0,2) | 60842145 |    0
 (0,3) | 60842146 |    0
 (0,4) | 60842147 |    0
```

```
SELECT ctid, xmin, xmax, chunkid from test_toast ;
```



t_xmin / t_xmax: These store the Transaction IDs (XIDs) that created and deleted/updated the row. This is the backbone of MVCC (Multi-Version Concurrency Control).
t_cid: The Command Identifier; it tracks which specific command within a transaction produced the tuple.
t_ctid: A pointer (Block ID and Offset) to the current physical location of the row. If the row was updated, this points to the new version.
t_infomask: A bitmask used to flag the row's status (e.g., has nulls, has variable-width attributes, or is "frozen").
t_hoff: The "Header Offset," which tells the system where the actual user data begins after the header and the Null Bitmap.

How to find toast data?

```
select
   lp,
   t_attrs
from heap_page_item_attrs(get_raw_page('public.test_toast', 0), 'public.test_toast') ;
```

Lets take a look at the TupleDescriptor

```
select
   lp,
   t_attrs
from heap_page_item_attrs(get_raw_page('public.test_toast', 0), 'public.test_toast') ;

 lp |  t_attrs
----+-----------------------------------------------------------------------------------
  1 | {"\\x1b73686f727420737472696e67"}
  2 | {"\\x900c0000383030206c656e6774682073747269e673a20554830343749373850345…."}
  3 | {"\\x0112c8090000c40900003e8e1200368e1200"}
  4 | {"\\x01128c130000881300003f8e1200368e1200"}
```

```
\x01128c1300008813000003f8e1200368e1200
```

```
\x
  01    header
    12    tag
      8c130000    original length
      88130000    stored length
      3f8e1200    chunk_id
      368e1200    toastrelid
```

```
\x
  01
    12
      8c130000
      88130000
      3f8e1200
      368e1200
```

```
SELECT ('x' || lpad(hex, 16, '0'))::bit(64)::bigint AS chunk_id
FROM  (
   VALUES
      ('00128e3f')
    ) t(hex);

 chunk_id
----------
  1216063
```

63 * 16^0 + 142 * 16^2 + 18 * 16^4 + 0 * 16^6 = 1216063

```
\x
  01
    12
      8c130000 -> 5004       original length
      88130000 -> 5000       stored length
      3f8e1200 -> 1216063    chunk_id
      368e1200 -> 1216054    toastrelid
```

```
select reltoastrelid
from pg_class
where relname = 'test_toast';

reltoastrelid
---------------
       1216054

select 1216054::regclass::text;

           text
---------------------------
 pg_toast.pg_toast_1216051
```

63 * 16^0 + 142 * 16^2 + 18 * 16^4 + 0 * 16^6 = 1216063

Difference between 5004 and 5000 is the 4 byte header.

```
\x
  01
    12
      8c130000 -> 5004
      88130000 -> 5000
      3f8e1200 -> 1216063
      368e1200 -> 1216054
```

```
select
  chunk_id,
  chunk_seq,
  octet_length(chunk_data)
from pg_toast.pg_toast_1216051
where chunk_id = 1216063;

 chunk_id | chunk_seq | octet_length
----------+-----------+--------------
  1216063 |         0 |         1996
  1216063 |         1 |         1996
  1216063 |         2 |         1008
```

63 * 16^0 + 142 * 16^2 + 18 * 16^4 + 0 * 16^6 = 1216063

Make this a group by grouping sets query

```
select



from
                    get_raw_page('public.test_toast', 0)
```

We need to use page inspect to find the chunk_id for a row.
Great blog about this topic: https://bdrouvot.wordpress.com/2020/01/04/get-toast-chunk_id-from-the-user-table-tuples-or-from-the-toast-index-thanks-to-pageinspect/

https://pgconf.ru/media/2016/05/13/tuple-internals.pdf

```
select

from
  heap_page_item_attrs(get_raw_page('public.test_toast', 0),'public.test_toast') as
  page_item_attrs ;
```

We need to use page inspect to find the chunk_id for a row.
Great blog about this topic: https://bdrouvot.wordpress.com/2020/01/04/get-toast-chunk_id-from-the-user-table-tuples-or-from-the-toast-index-thanks-to-pageinspect/

https://pgconf.ru/media/2016/05/13/tuple-internals.pdf

```
select


       page_item_attrs.t_attrs[1]


from
  heap_page_item_attrs(get_raw_page('public.test_toast', 0),'public.test_toast') as
  page_item_attrs ;
```

We need to use page inspect to find the chunk_id for a row.
Great blog about this topic: https://bdrouvot.wordpress.com/2020/01/04/get-toast-chunk_id-from-the-user-table-tuples-or-from-the-toast-index-thanks-to-pageinspect/

https://pgconf.ru/media/2016/05/13/tuple-internals.pdf

```
select


     substr(
       page_item_attrs.t_attrs[1],octet_length(page_item_attrs.t_attrs[1])
   -7,4)::text


from
  heap_page_item_attrs(get_raw_page('public.test_toast', 0),'public.test_toast') as
  page_item_attrs ;
```

We need to use page inspect to find the chunk_id for a row.
Great blog about this topic: https://bdrouvot.wordpress.com/2020/01/04/get-toast-chunk_id-from-the-user-table-tuples-or-from-the-toast-index-thanks-to-pageinspect/

https://pgconf.ru/media/2016/05/13/tuple-internals.pdf

```
select



   substr(
     substr(
       page_item_attrs.t_attrs[1],octet_length(page_item_attrs.t_attrs[1])
   -7,4)::text,3
  )

from
  heap_page_item_attrs(get_raw_page('public.test_toast', 0),'public.test_toast') as
  page_item_attrs ;
```

We need to use page inspect to find the chunk_id for a row.
Great blog about this topic: https://bdrouvot.wordpress.com/2020/01/04/get-toast-chunk_id-from-the-user-table-tuples-or-from-the-toast-index-thanks-to-pageinspect/

https://pgconf.ru/media/2016/05/13/tuple-internals.pdf

```
select


  regexp_replace(                                  12345678 -> 78563412
    substr(
      substr(
        page_item_attrs.t_attrs[1],octet_length(page_item_attrs.t_attrs[1])
    -7,4)::text,3
  ),'(\w\w)(\w\w)(\w\w)(\w\w)','\4\3\2\1')

from
  heap_page_item_attrs(get_raw_page('public.test_toast', 0),'public.test_toast') as
  page_item_attrs ;
```

We need to use page inspect to find the chunk_id for a row.
Great blog about this topic: https://bdrouvot.wordpress.com/2020/01/04/get-toast-chunk_id-from-the-user-table-tuples-or-from-the-toast-index-thanks-to-pageinspect/

https://pgconf.ru/media/2016/05/13/tuple-internals.pdf

```
select

 (‘x'||
  regexp_replace(
    substr(
      substr(
        page_item_attrs.t_attrs[1],octet_length(page_item_attrs.t_attrs[1])
    -7,4)::text,3
  ),’(\w\w)(\w\w)(\w\w)(\w\w)’,'\4\3\2\1')
 )::bit(32)::bigint as chunk_id
from
  heap_page_item_attrs(get_raw_page('public.test_toast', 0),'public.test_toast') as
  page_item_attrs ;
```

We need to use page inspect to find the chunk_id for a row.
Great blog about this topic: https://bdrouvot.wordpress.com/2020/01/04/get-toast-chunk_id-from-the-user-table-tuples-or-from-the-toast-index-thanks-to-pageinspect/

https://pgconf.ru/media/2016/05/13/tuple-internals.pdf

```
select
lp,
 (‘x'||
  regexp_replace(
    substr(
      substr(
        page_item_attrs.t_attrs[1],octet_length(page_item_attrs.t_attrs[1])
    -7,4)::text,3
  ),’(\w\w)(\w\w)(\w\w)(\w\w)’,'\4\3\2\1')
 )::bit(32)::bigint as chunk_id
from
  heap_page_item_attrs(get_raw_page('public.test_toast', 0),'public.test_toast') as
  page_item_attrs ;
```

We need to use page inspect to find the chunk_id for a row.
Great blog about this topic: https://bdrouvot.wordpress.com/2020/01/04/get-toast-chunk_id-from-the-user-table-tuples-or-from-the-toast-index-thanks-to-pageinspect/

https://pgconf.ru/media/2016/05/13/tuple-internals.pdf

```
 lp |  chunk_id
----+------------
  1 | 1953702004
  2 |  928403784
  3 |    1216062
  4 |    1216063
```

This looks nice, but not all data is actually toasted.

Mental break

```
select
   lp,
   t_attrs
from heap_page_item_attrs(get_raw_page('public.test_toast', 0), 'public.test_toast') ;

 lp |  t_attrs
----+-----------------------------------------------------------------------------------
  1 | {"\\x1b73686f727420737472696e67"}
  2 | {"\\x900c0000383030206c656e6774682073747269e673a20554830343749373850345…."}
  3 | {"\\x0112c8090000c40900003e8e1200368e1200"}
  4 | {"\\x01128c130000881300003f8e1200368e1200"}
```

```
select
      lp,
      get_bit(t_attrs[1], 0) as header_bit,
      get_byte(t_attrs[1], 0) as header_byte
from heap_page_item_attrs(get_raw_page('test_toast', 0), 'test_toast'::regclass);
```

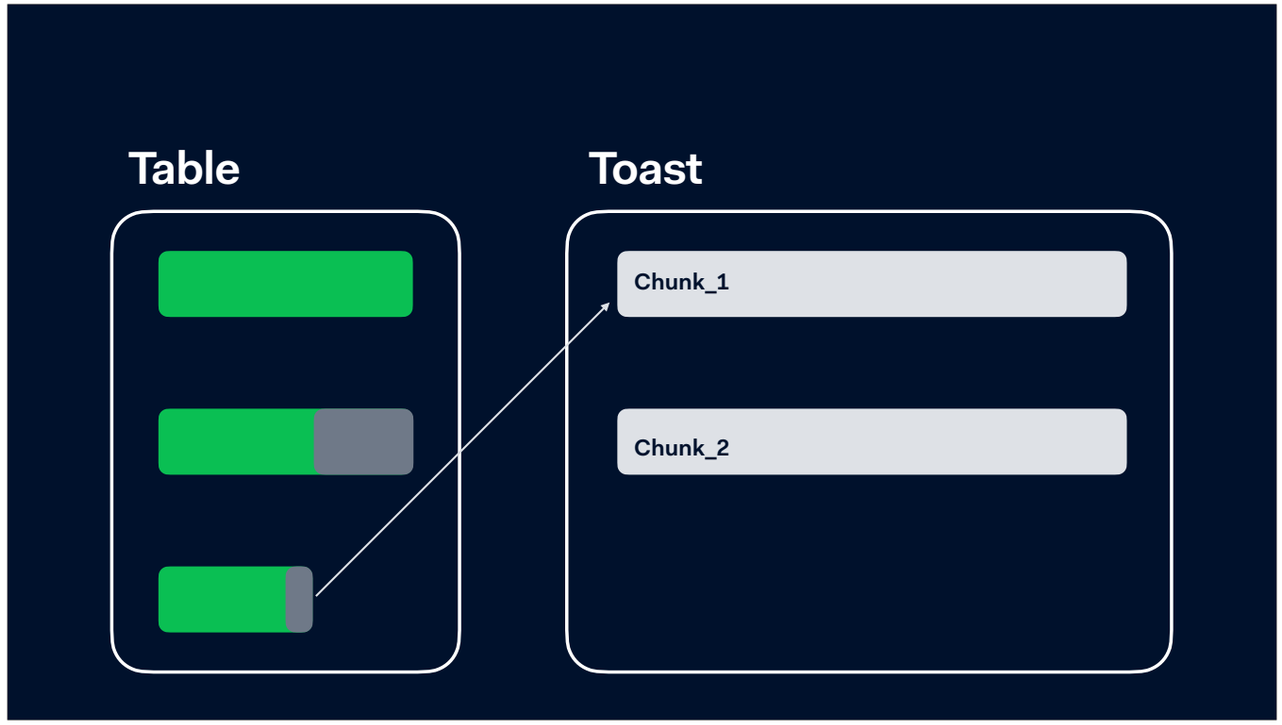
```
select lp,length(t_attrs[1]),
       substr(t_attrs[1], 1, 1),
       case when get_bit(t_attrs[1], 0) = 1 then 'short' else 'normal-size' end,
       case when get_byte(t_attrs[1], 0) = 1 then
         get_byte(t_attrs[1], 10) +
         (get_byte(t_attrs[1], 11) << 8) +
         (get_byte(t_attrs[1], 12) << 16) +
         (get_byte(t_attrs[1], 13) << 24)
       else 0 end as "toast chunk ID"
from heap_page_item_attrs(get_raw_page('test_toast', 0), 'test_toast'::regclass);
```

Combining it all and use bit shift instead of regex replace
This query actually checks whether the data is in external toast table.

```
 lp | length | varlena type | toast chunk ID
----+--------+--------------+----------------
  1 |     13 | short        |              0
  2 |    804 | normal-size  |              0
  3 |     18 | short        |        1216062
  4 |     18 | short        |        1216063
```

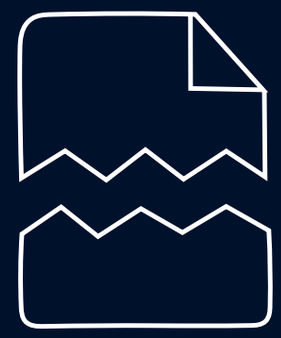
What did we learn this far

We kunnen nu van tabel naar toast tabel springen. Als je hier verdwaald was, onthoud dan alleen dat.



Uitleg over concepten toast.
Een chunk kan uit meerdere sequences bestaan.

Problem



```
37250763372 XX001   missing chunk number 0 for toast value 4083590432 in pg_toast_36822
37250832089 XX001   unexpected chunk number 2 (expected 0) for toast value 4083621490 in pg_toast_36822
```

Problem

```
INFO:  aggressively vacuuming "pg_toast.pg_toast_1216051"
ERROR:  found xmin 2708558663 from before relfrozenxid 2960707532
CONTEXT:  while scanning block 0 of relation "pg_toast.pg_toast_1216051"
```

The first sign of the problem: failing vacuum.

Problem

```
select * from pg_visibility('pg_toast.pg_toast_1216051',0);
 all_visible | all_frozen | pd_all_visible
-------------+------------+----------------
 t           | t          | f
```

Check the visibility map whether all rows are visible
all_visible and all_frozen are visibility map flags. pd_all_visible is the all visibility flag from the page itself.

Problem

```
select lp, xmin from heap_page_items(get_raw_page('pg_toast.pg_toast_1216051',0));
lp  |   xmin
—+————————-+
  1 | 2708558663
```

Use page inspect to check the page.

Problem

```
select * from test_toast where id = 1;
ERROR:  could not access status of transaction 2709439393
DETAIL:  Could not open file "pg_xact/0A17": No such file or directory.
```

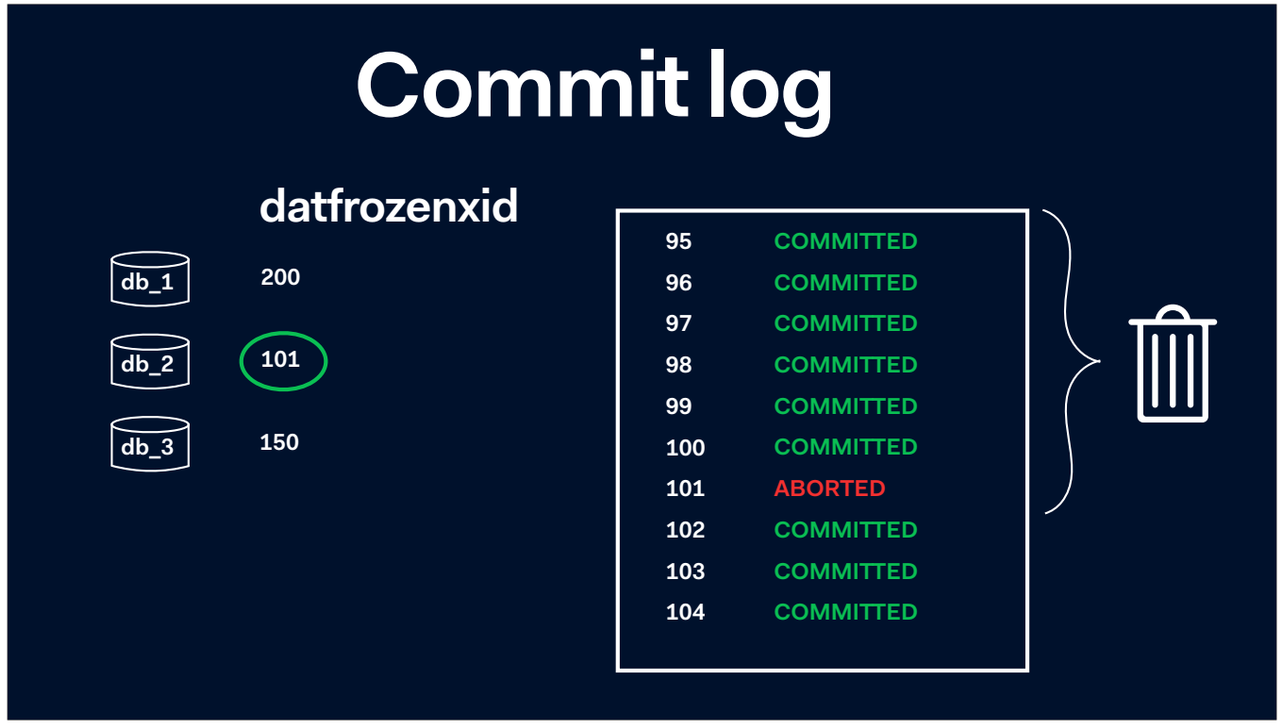
Try to select all data from the page.

The file pg_xact/0A17 depends on the transaction number and some logic.

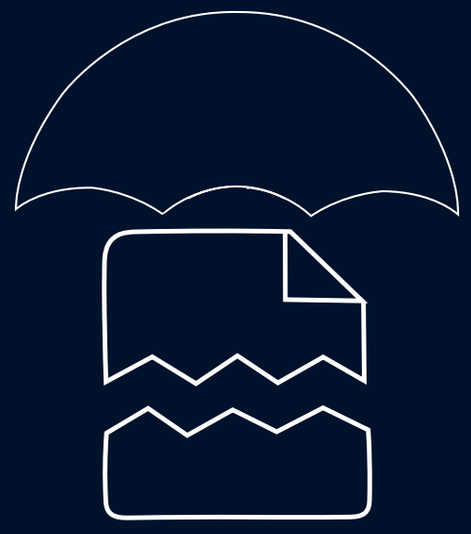
Commit log

100	COMMITTED
101	ABORTED
102	COMMITTED
103	COMMITTED
104	IN_PROGRESS
105	IN_PROGRESS
106	IN_PROGRESS
107	IN_PROGRESS
108	COMMITTED
109	IN_PROGRESS

https://www.interdb.jp/pg/pgsql05/04.html#543-maintenance-of-the-clog
Commit log contains the transaction status. IN_PROGRESS, COMMITTED, ABORTED, and SUB_COMMITTED.
When PostgreSQL shuts down or whenever the checkpoint process runs, the data of the clog are written into files stored in the pg_xact subdirectory


Commit log
datfrozenxid
db_1
200
db_2
101
db_3
150
95 COMMITTED
96 COMMITTED
97 COMMITTED
98 COMMITTED
99 COMMITTED
100 COMMITTED
101 ABORTED
102 COMMITTED
103 COMMITTED
104 COMMITTED

Mitigation



```
select * from test_toast where id = 1;
ERROR:  could not access status of transaction 2709439393
DETAIL:  Could not open file "pg_xact/0A17": No such file or directory.
```

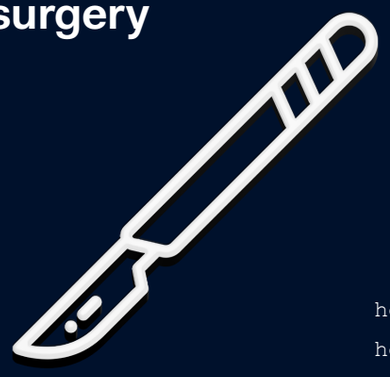
Try to select all data from the page.

The file pg_xact/0A17 depends on the transaction number and some logic.

Create a `pg_xact` file with all-committed transactions

Try to create a clog with all transactions marked as being committed.
Didn't work, page is still a mess.

pg_surgery



heap_force_kill

heap_force_freeze

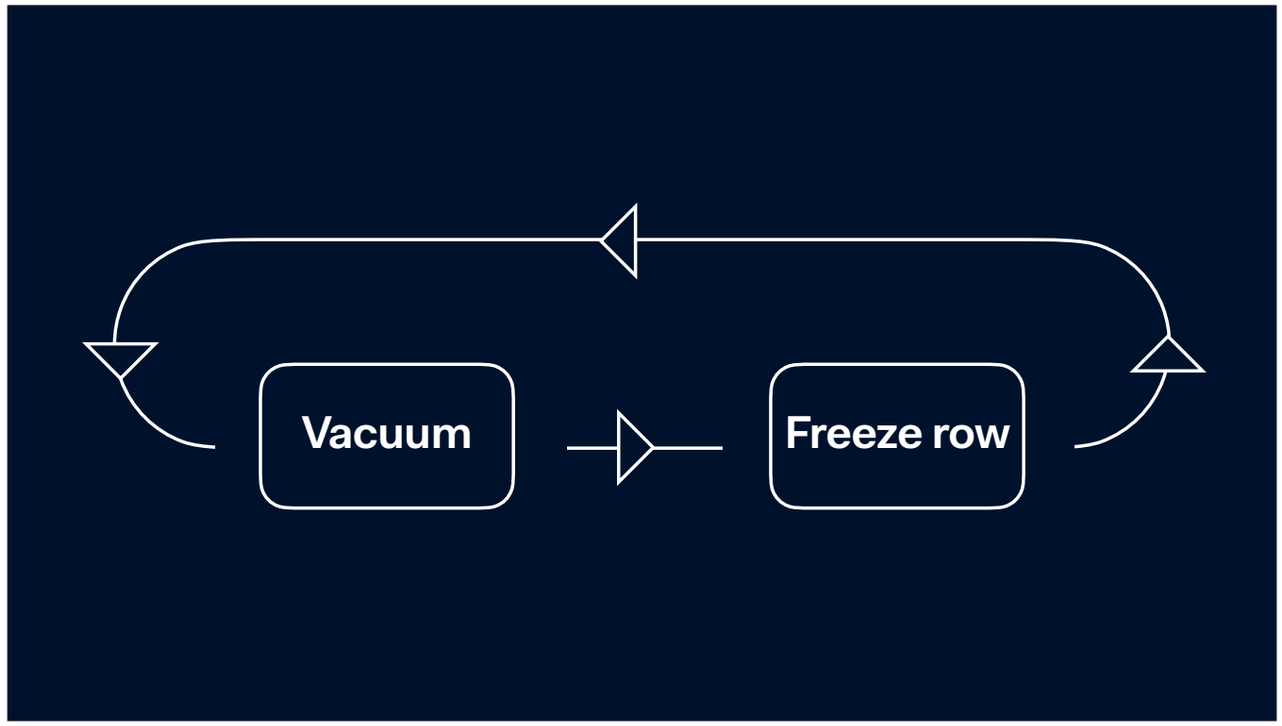
Heap_force_kill marks the line pointer as being dead. (Set status on lp_flag to 3 (source itemid.h))

Heap_force_freeze set the freeze bits on the row.

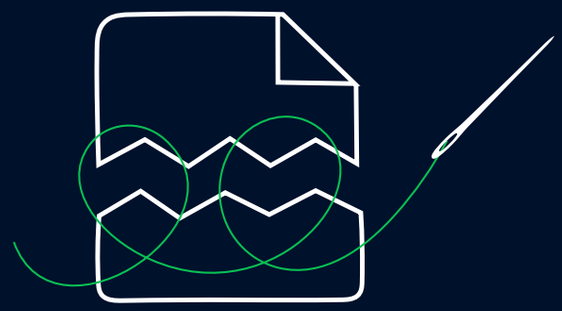
Hacker list discussion on the pg_surgery extension: https://www.postgresql.org/message-id/flat/CA%2BTgmoZW1fsU-QUNCRUQMGUygBDPVeOTLCqRdQZch%3DEYZnctSA%40mail.gmail.com

```
create extension pg_surgery ;
```

Try to create a clog with all transactions marked as being committed.
Didn't work, page is still a mess.


Vacuum
Freeze row

Solution



Important questions

1. Why did this suddenly happen?
2. Did we lose any data?

First important question: did we loose any data???

Problem definition #1

- Pages marked as all visible in visibility map
- Heap pages have not been ‘cleaned’

Every solution starts with a good problem definition
What is the actual problem?
Until now we have been looking at single page

We can fix the page, but index might still be corrupted.
Heap should have been cleanup by vacuum of single page cleanup.

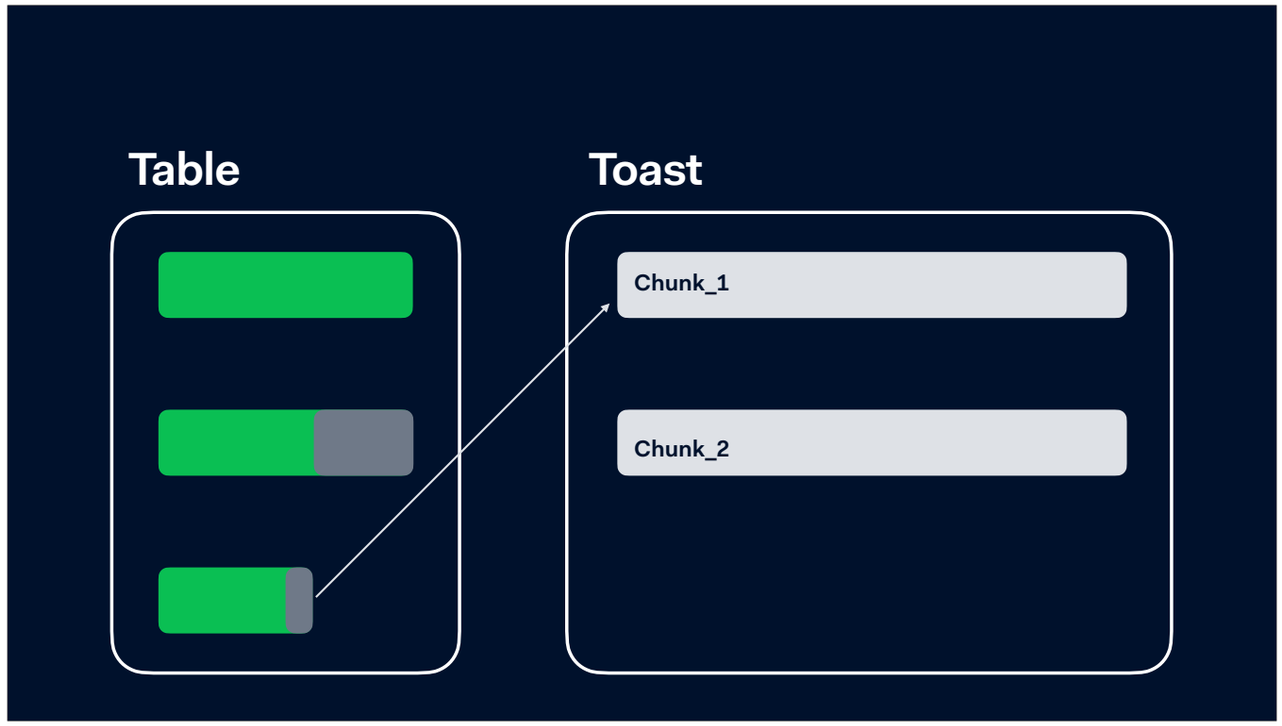
Problem definition #2

- Invalid x_min
- Invalid x_max
- missing chunk number 0 for toast value
- unexpected chunk number 2 (expected 0) for toast value

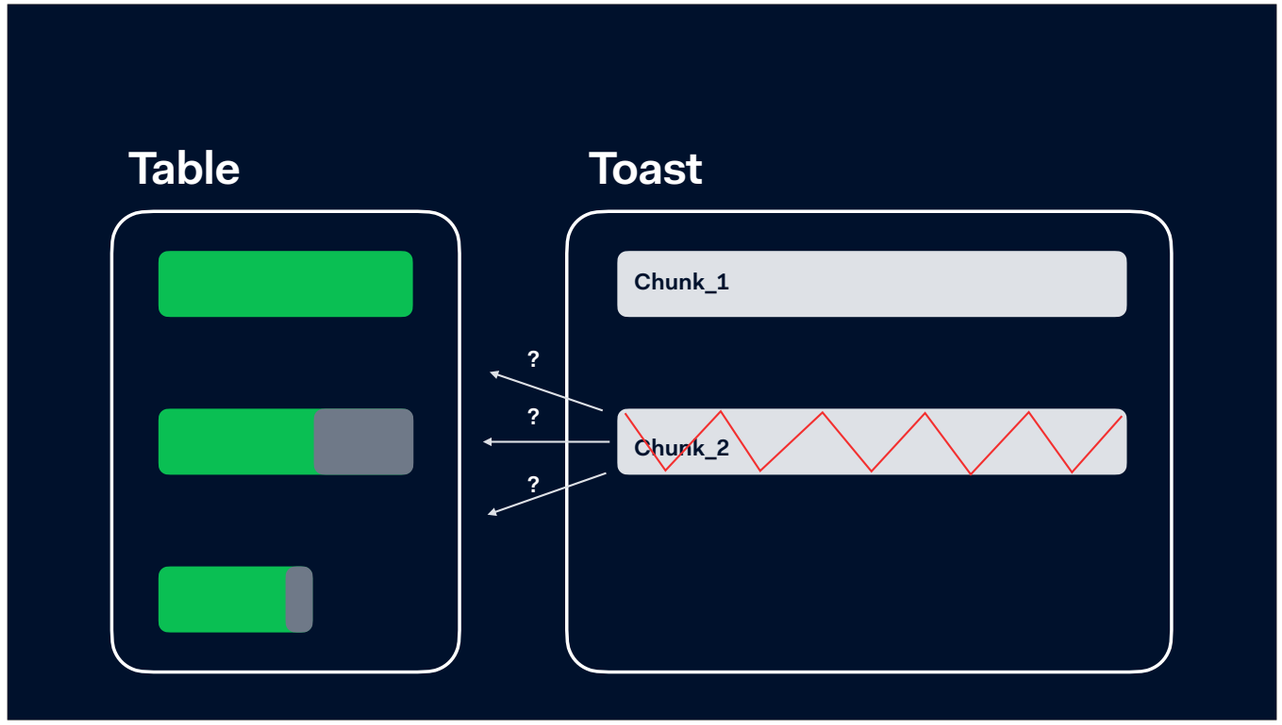
Problem definition #3

Corruptions in toast

- heap
- index



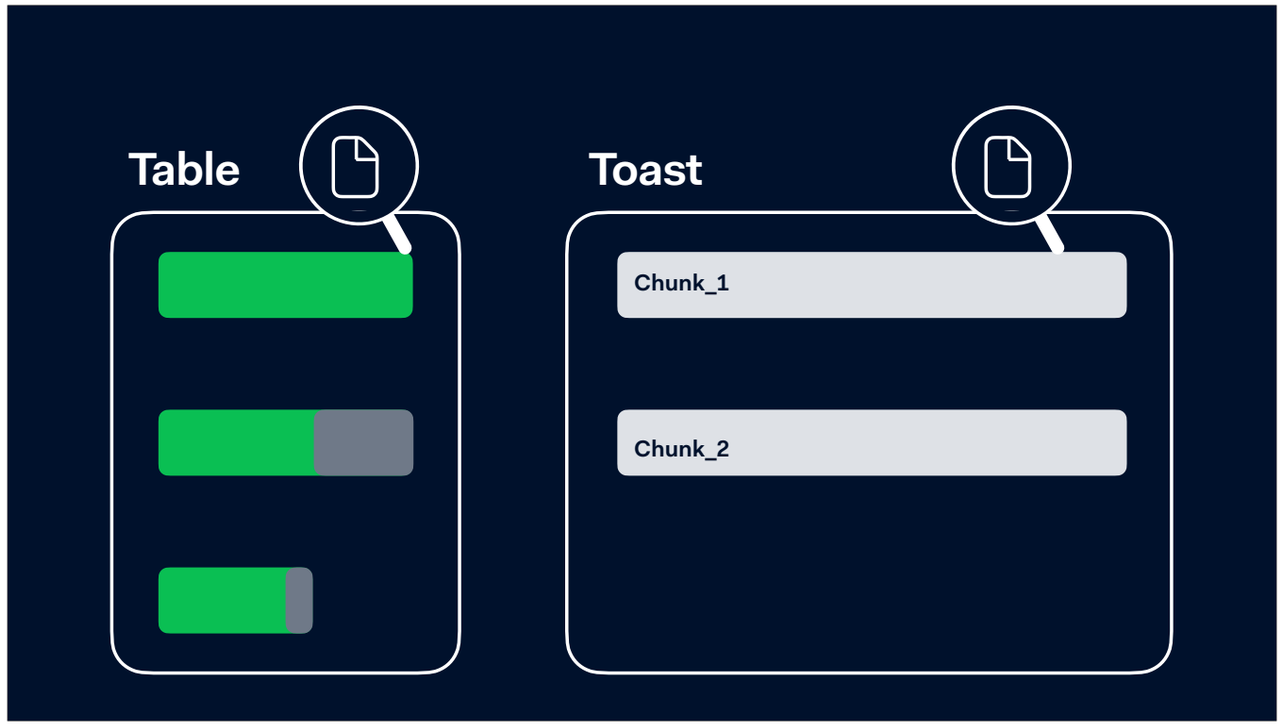
From table to toast is easy. And for the main table I know the transaction status



From toast to table back is much harder.

Run pg_check_frozen from the pg_visibility extension

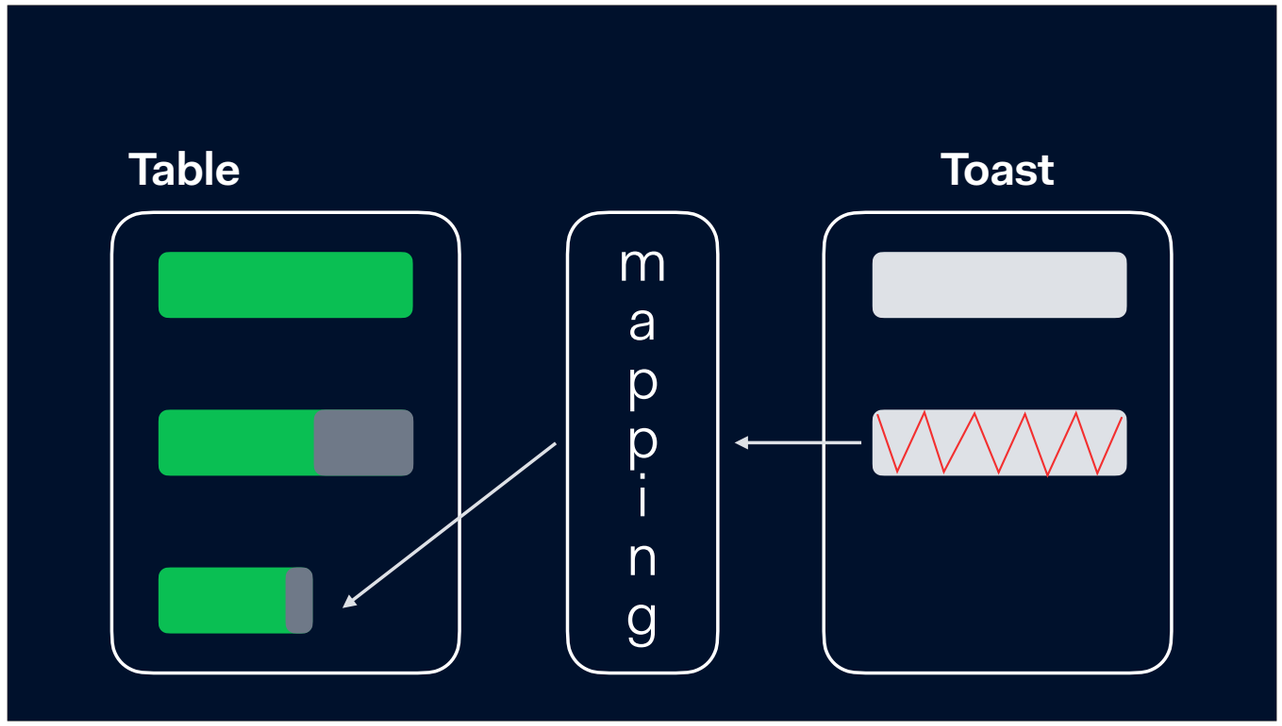
Returns the TIDs of non-frozen tuples stored in pages marked all-frozen in the visibility map. If this function returns a non-empty set of TIDs, the visibility map is corrupt.



How do we build a map?

Use page inspect to collect all chunk_id's from the main table

Use page_inspect to get the chunk_id's from the toast table. Just using queries doesn't work, because the data is corrupted and therefor 'not visible' to end user using psql.



From toast to table back is much harder.

No data loss

All corruptions originate from the two days before the major upgrade

Toast data exists

Transaction status is unclear

Transaction status

```
select


from
  heap_page_items( get_raw_page('pg_toast.pg_toast_1216051', 0) ) hpi,
```

In order to fix the data, we need to understand the transaction status, which is written in t_infomask and can be queried using page_inspect

```
select



from
  heap_page_items( get_raw_page('pg_toast.pg_toast_1216051', 0) ) hpi,
  LATERAL heap_tuple_infomask_flags(t_infomask, t_infomask2) hti ;
```

In order to fix the data, we need to understand the transaction status, which is written in t_infomask and can be queried using page_inspect

```
select
  lp,
  hti.raw_flags,
  combined_flags
from
  heap_page_items( get_raw_page('pg_toast.pg_toast_1216051', 0) ) hpi,
  LATERAL heap_tuple_infomask_flags(t_infomask, t_infomask2) hti ;
```

In order to fix the data, we need to understand the transaction status, which is written in t_infomask and can be queried using page_inspect

HEAP_HASVARWIDTH	Has Variable-width attributes
HEAP_XMIN_COMMITTED	t_xmin committed
HEAP_XMIN_INVALID	t_xmin invalid / aborted
HEAP_XMAX_COMMITTED	t_xmax committed
HEAP_XMAX_INVALID	t_xmax invalid / aborted
HEAP_KEYS_UPDATED	tuple was updated and key cols modified, or tuple deleted
HEAP_XMIN_FROZEN	(HEAP_XMIN_COMMITTED|HEAP_XMIN_INVALID)

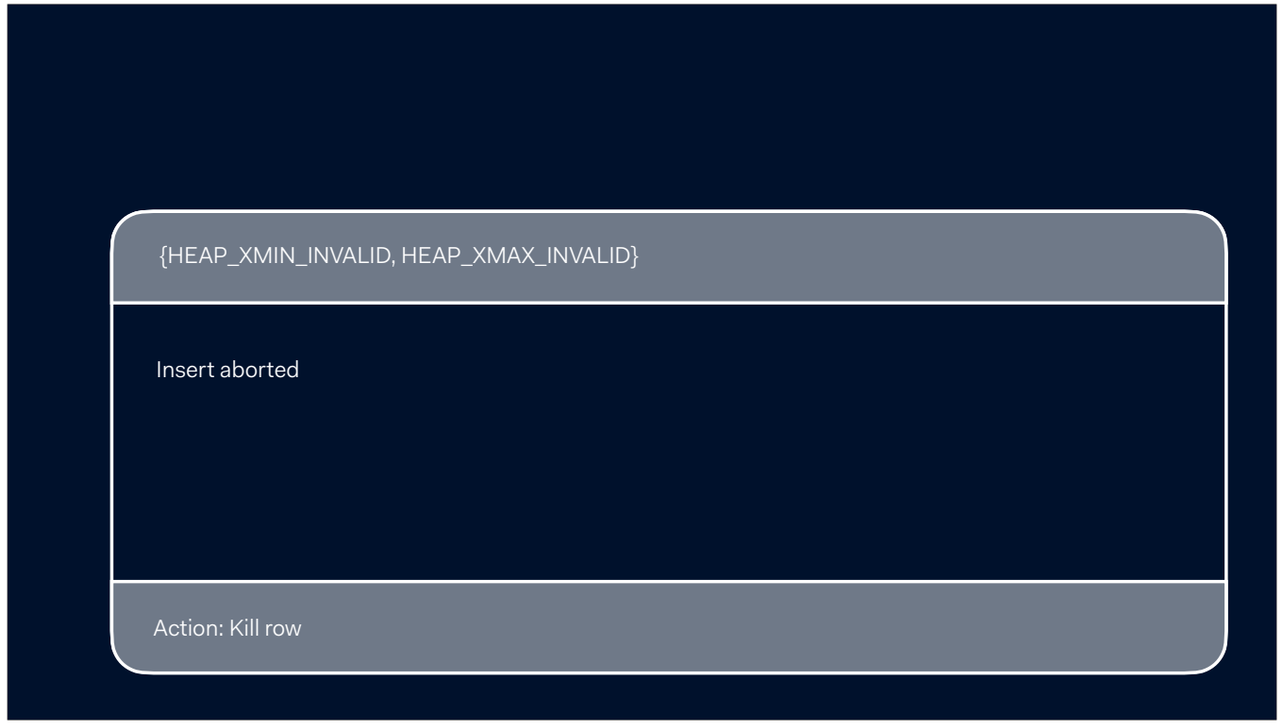
src/include/access/htup_details.h

```
/*
 * information stored in t_infomask:
 */
#define HEAP_HASNULL                    0x0001          /* has null attribute(s) */
#define HEAP_HASVARWIDTH                0x0002          /* has variable-width attribute(s) */
#define HEAP_XMIN_COMMITTED                     0x0100          /* t_xmin committed */
#define HEAP_XMIN_INVALID               0x0200          /* t_xmin invalid/aborted */
#define HEAP_XMIN_FROZEN                (HEAP_XMIN_COMMITTED|HEAP_XMIN_INVALID)
#define HEAP_XMAX_COMMITTED                     0x0400          /* t_xmax committed */
#define HEAP_XMAX_INVALID               0x0800          /* t_xmax invalid/aborted */
#define HEAP_KEYS_UPDATED               0x2000          /* tuple was updated and key cols modified, or tuple deleted */
#define HEAP_XMIN_FROZEN                (HEAP_XMIN_COMMITTED|HEAP_XMIN_INVALID)
```

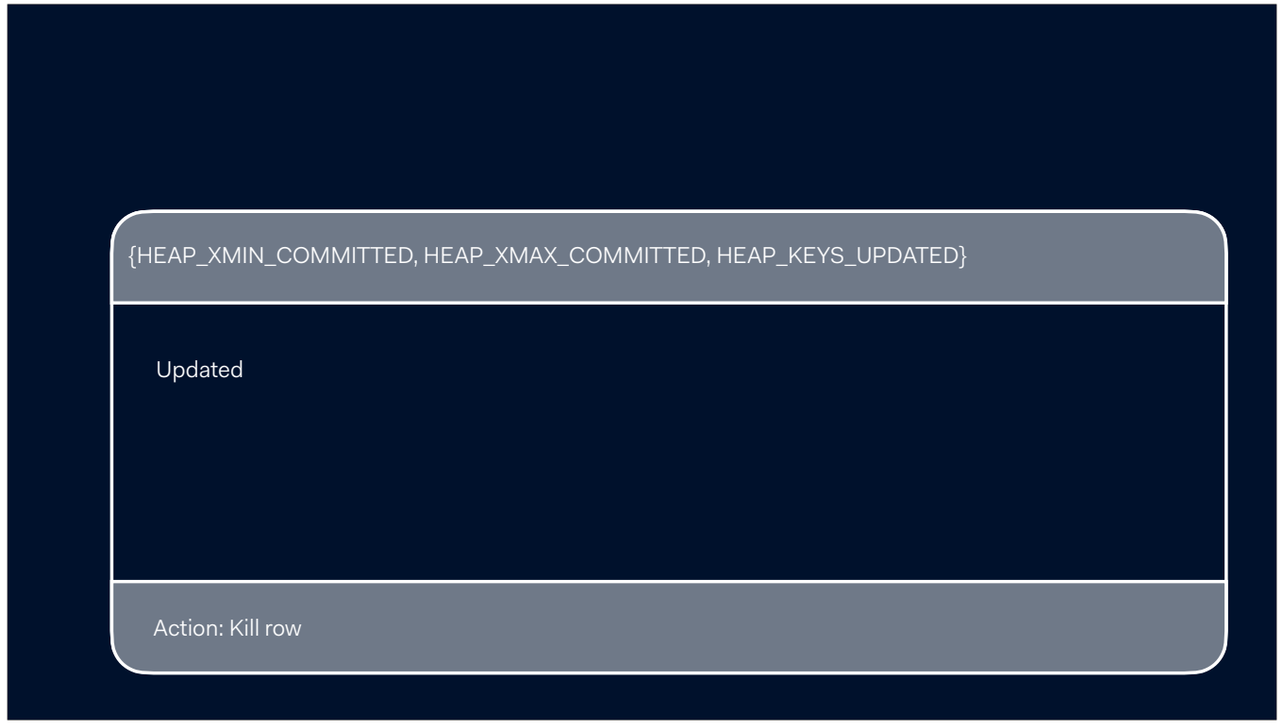
https://techsupport.enterprisedb.com/rt/ticket/90633/?correspondence=833525

https://techsupport.enterprisedb.com/rt/ticket/90633/?correspondence=833525

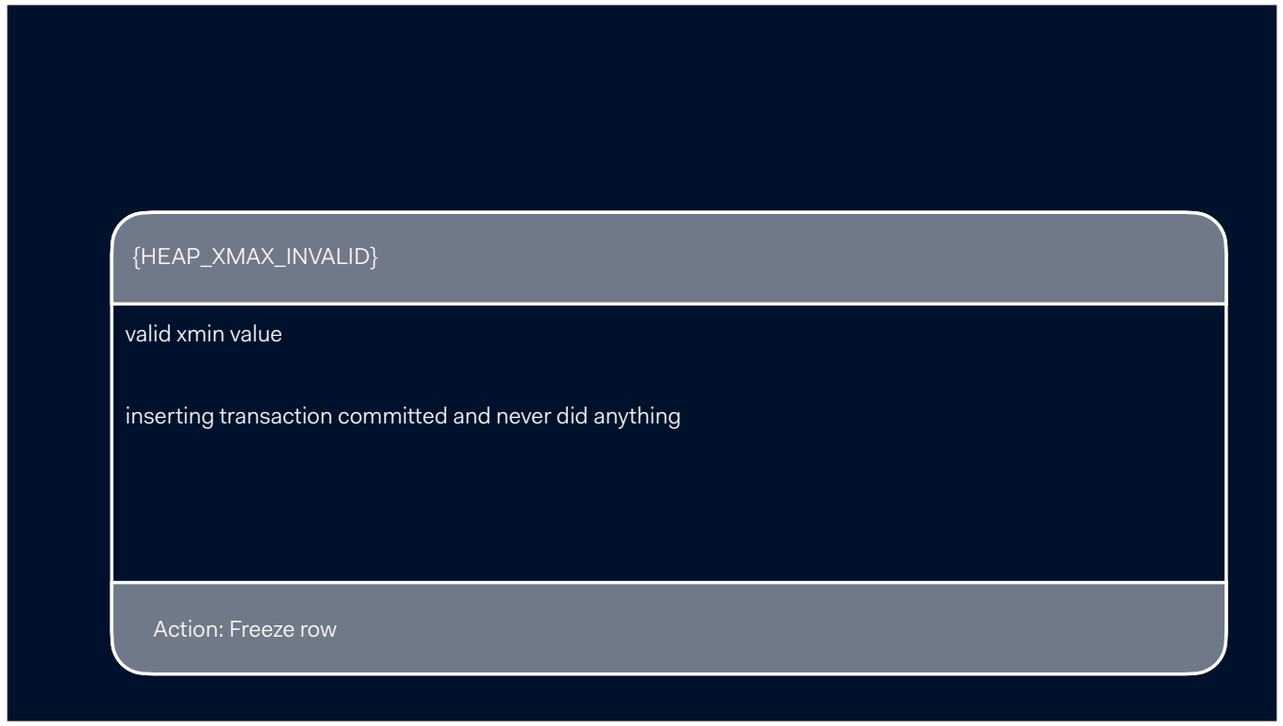
https://techsupport.enterprisedb.com/rt/ticket/90633/?correspondence=833525



https://techsupport.enterprisedb.com/rt/ticket/90633/?correspondence=833525



https://techsupport.enterprisedb.com/rt/ticket/90633/?correspondence=833525



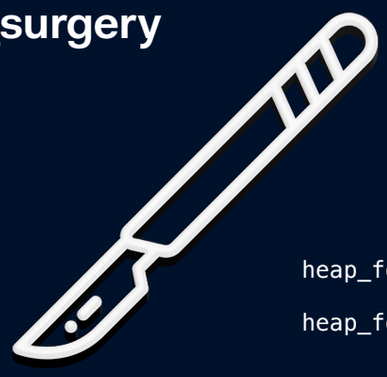
https://techsupport.enterprisedb.com/rt/ticket/90633/?correspondence=835357

Heh, yeah, "never did anything" is not fully correct. What I meant is
that this tuple doesn't have HEAP_XMIN_COMMITTED, which means we don't
know if the transaction committed or not. Going by my comments on the
ticket update I just made in response to Derk, it is quite possible that
this tuple is invalid: if any other transaction had scanned this page,
it would have set HEAP_XMIN_COMMITTED and we would not be having this
discussion; we'd know to set it frozen. The most likely guess is that
the page was indeed visited by other transactions, and because the
inserting transaction had aborted, then they didn't set XMIN_COMMITTED.
So the tuple is likely dead.

HOWEVER, if we do guess that way, and are wrong, then we have lost data.
If we guess the other way, and are wrong, the worst that happens is that
we have a lingering TOAST tuple and mostly no harm is done.

It's better (less risky) to be wrong in the way that causes the least
damage.

pg_surgery



`heap_force_kill`

`heap_force_freeze`

Summary

What did we learn this far

We kunnen nu van tabel naar toast tabel springen. Als je hier verdwaald was, onthoud dan alleen dat.

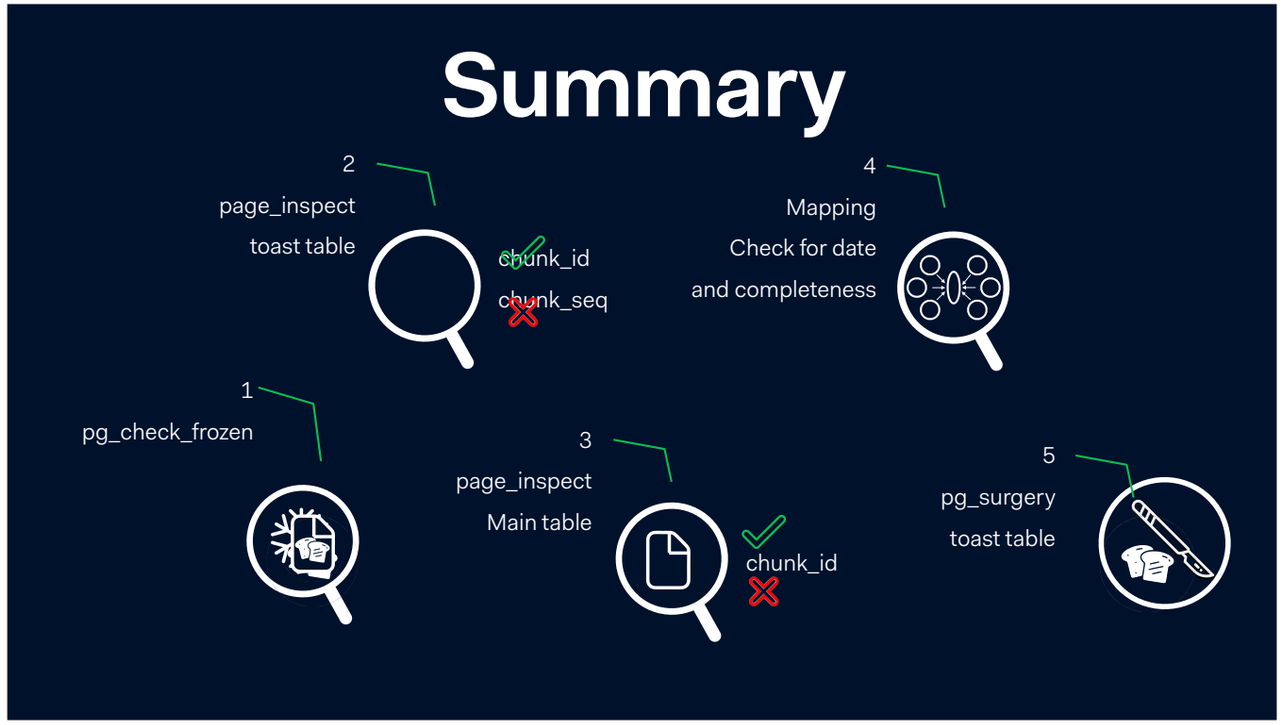
Summary

Toast is not trivial to understand

Extensions

- pageinspect
- pg_visibility
- pg_surgery
- pg_check_frozen

Summary


2
page_inspect
toast table
chunk_id
chunk_seq
4
Mapping
Check for date
and completeness
1
pg_check_frozen
3
page_inspect
Main table
chunk_id
5
pg_surgery
toast table

Lessons learned

Detecting

- pg_check_frozen
- amrepair (indexes)

Lessons learned

Prevention

- Run rsync without `--size-only` for vm and fsm
- Finish all vacuum before upgrade
- Run Checkpoint before upgrade

Lessons learned

Don’t do things on a live system unless you **fully understand** what you are doing

Special thanks

Cagri Biroglu

Álvaro Herrera Muñoz

Article about corruptions