Detecting and fixing corruptions

adyen

engineered
for ambition

Introduction story:

I am afraid of three things.
Messing up production data
Not being able to solve a production issue
Corruptions

This deck contains multiple ways to corrupt your database

This deck contains multiple ways to corrupt your database

Be very careful to use this on a live database

This deck contains multiple ways to corrupt your database

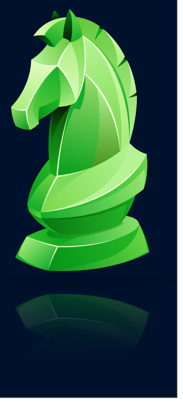
Be very careful to use this on a live database

and don’t blame me when you screw up

Derk van Veen

Database specialist

Adyen



adyen

engineered
for ambition

Agenda

Toast

Problem

Mitigation

Solution

Lessons learned

The next best thing after sliced bread.

The Oversized Attribute Storage Technique

Attributes are table columns in the postgres context

3 options for toastable data

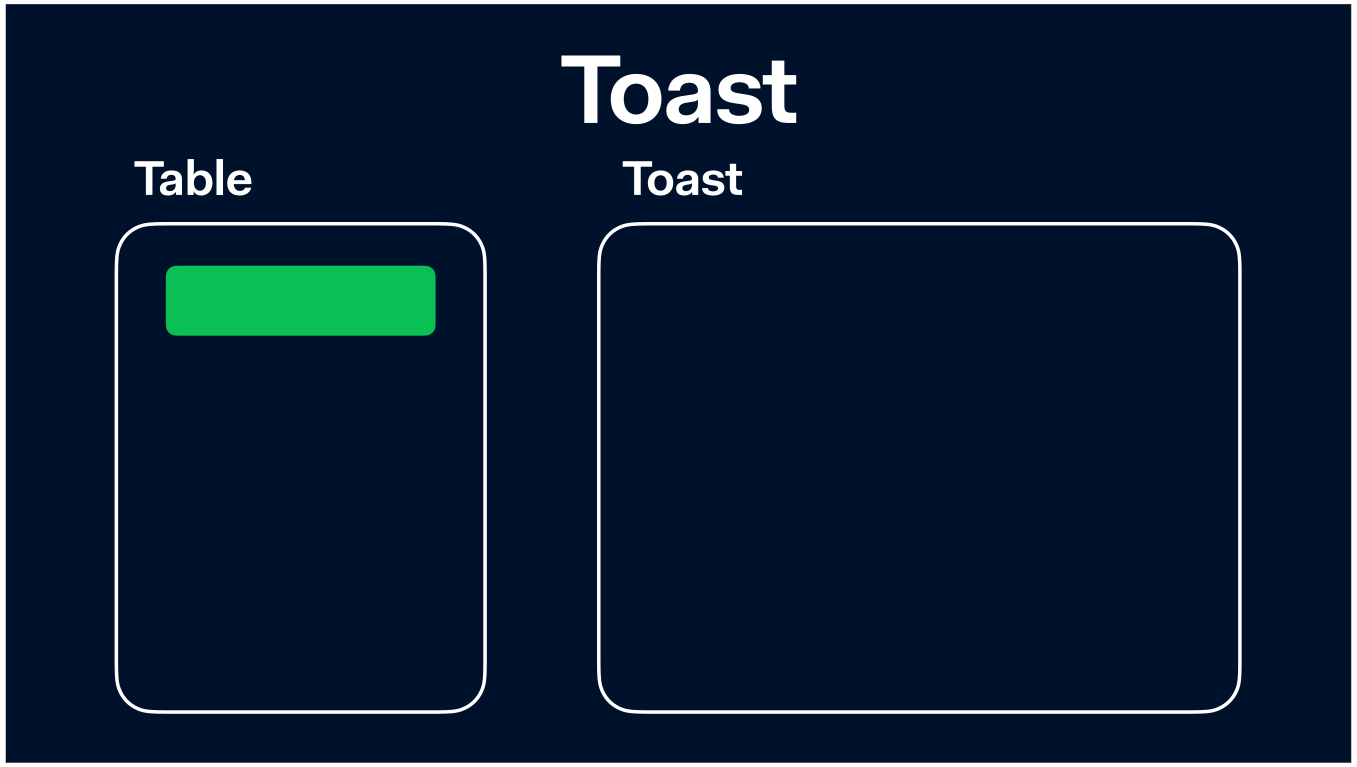
- Inline

3 options for toastable data

- Inline
- Compressed in line

3 options for toastable data

- Inline
- Compressed in line
- **Extended**

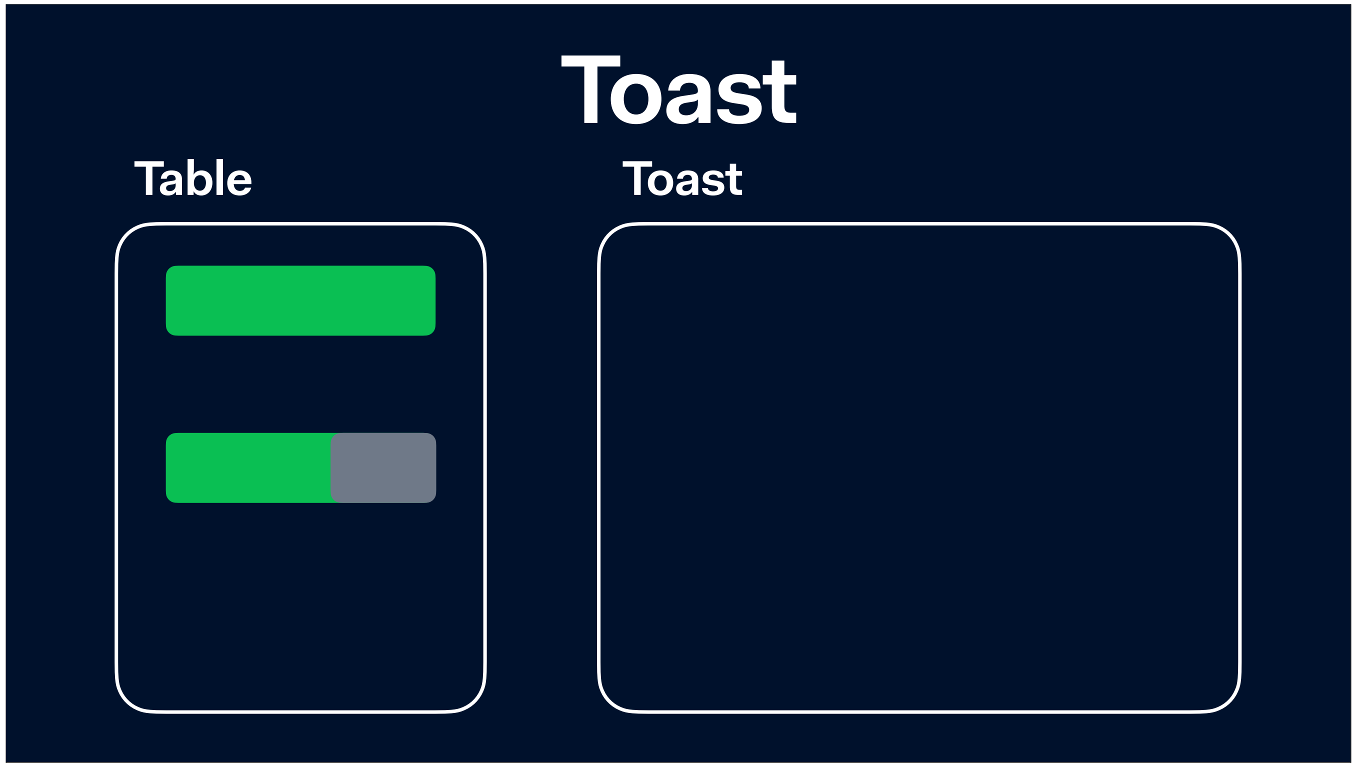


Toast

Table

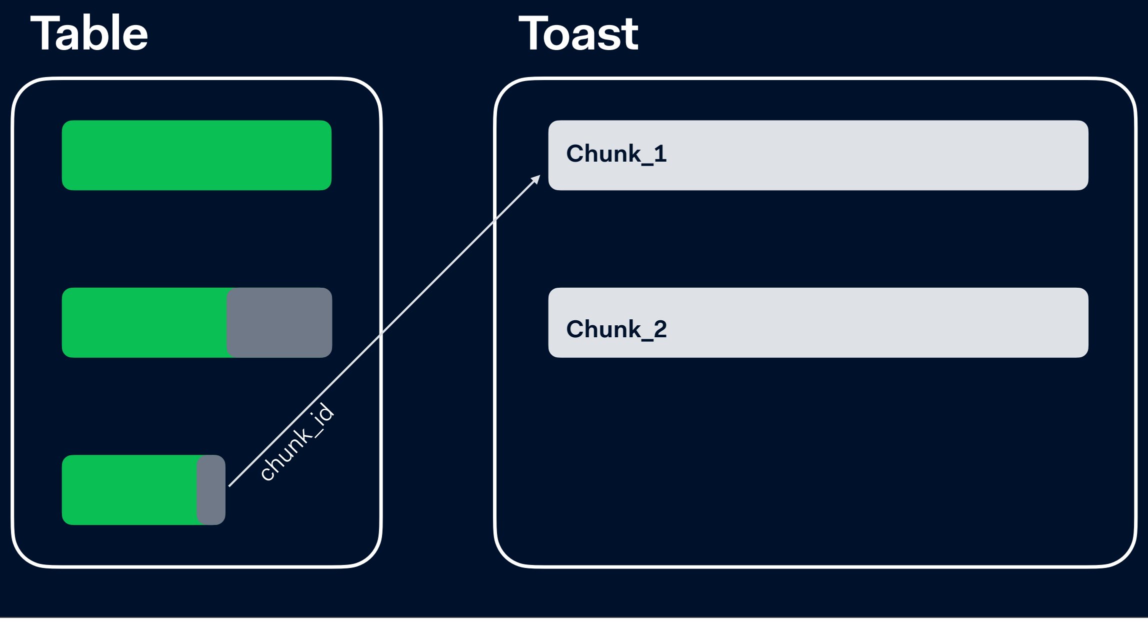
Toast

Only the third row contains data in the toast table.
Primary key for toast is chunk_id. A chunk might consists of multiple sequences.



Toast

Toast



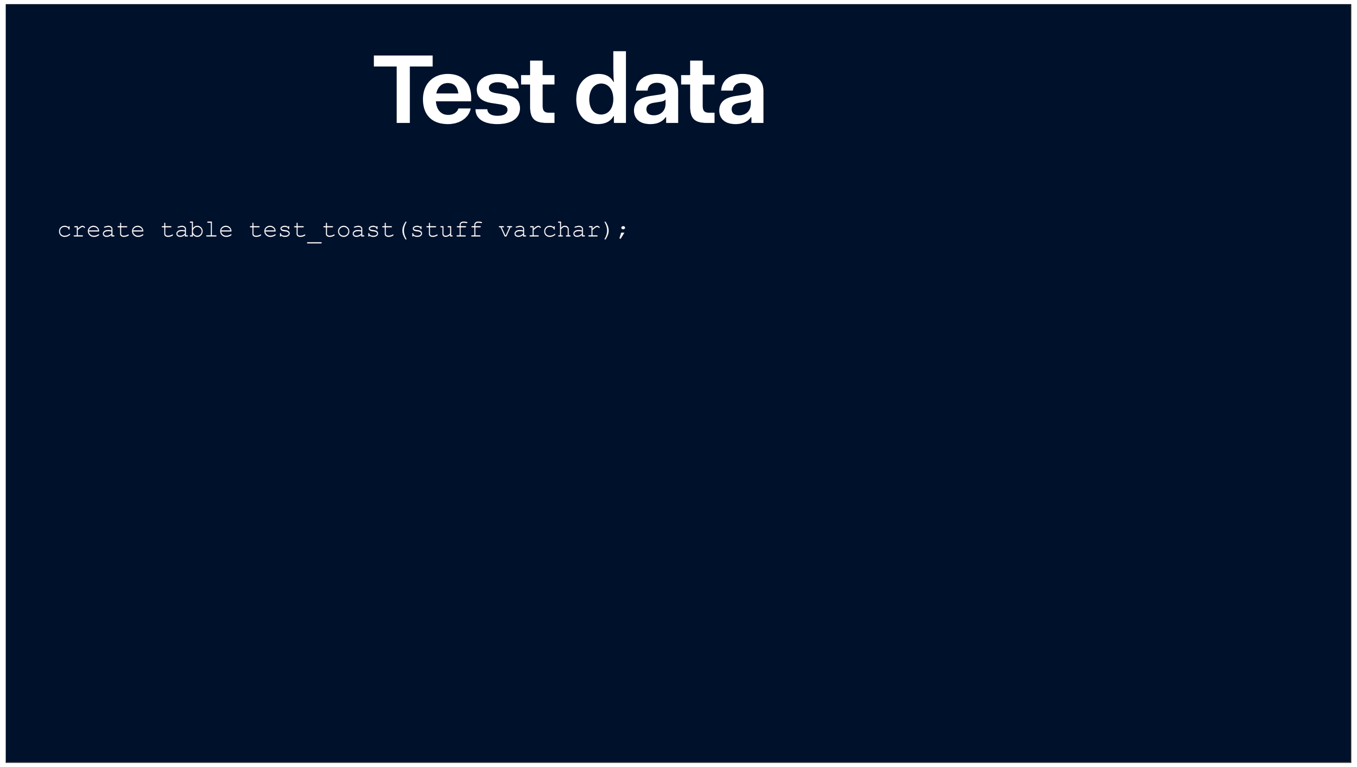
Table

Toast

Chunk_1

Chunk_2

chunk_id



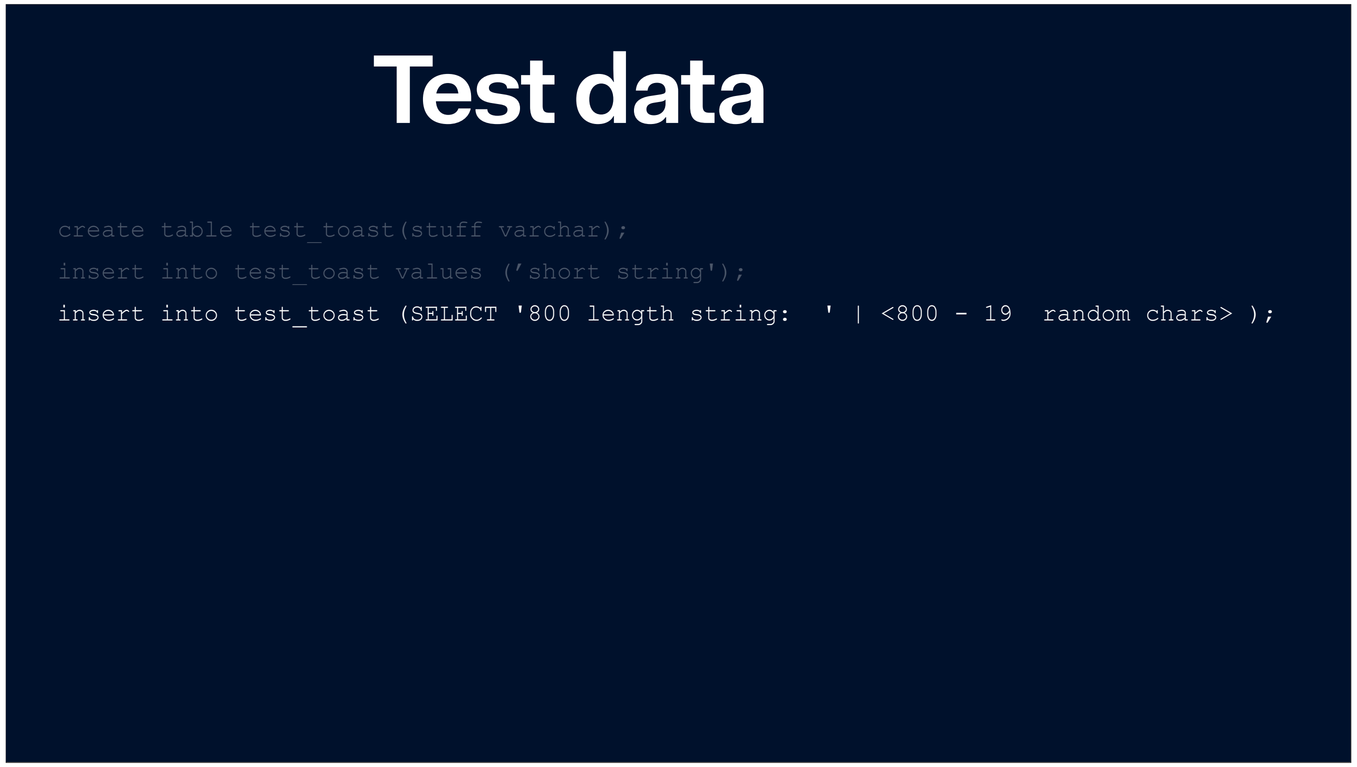
Test data

```
create table test_toast(stuff varchar);
```

```
create table test_toast(stuff varchar);
insert into test_toast values ('short string');
insert into test_toast (SELECT '800 length string: ' || array_to_string(array(select substr('ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 ',((random()*(37-1)+1)::integer),1) from generate_series(1,800 - 19)),''));
insert into test_toast (SELECT '2500 length string: ' || array_to_string(array(select substr('ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 ',((random()*(37-1)+1)::integer),1) from generate_series(1,2500 - 20)),''));
insert into test_toast (SELECT '5000 length string: ' || array_to_string(array(select substr('ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 ',((random()*(37-1)+1)::integer),1) from generate_series(1,5000 - 20)),''));
```

Test data

```
create table test_toast(stuff varchar);
insert into test_toast values (’short string');
```

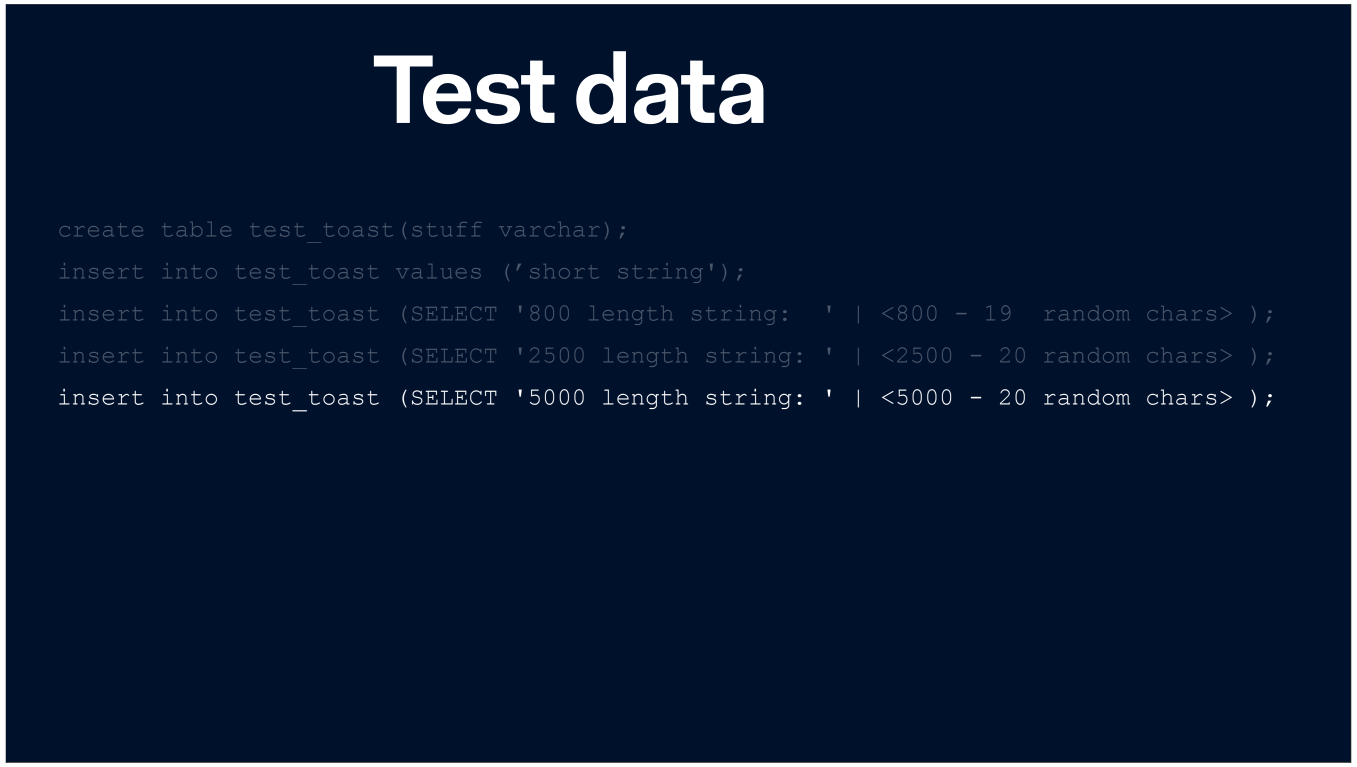


Test data

```
create table test_toast(stuff varchar);
insert into test_toast values (’short string');
insert into test_toast (SELECT '800 length string:  ' | <800 - 19  random chars> );
```

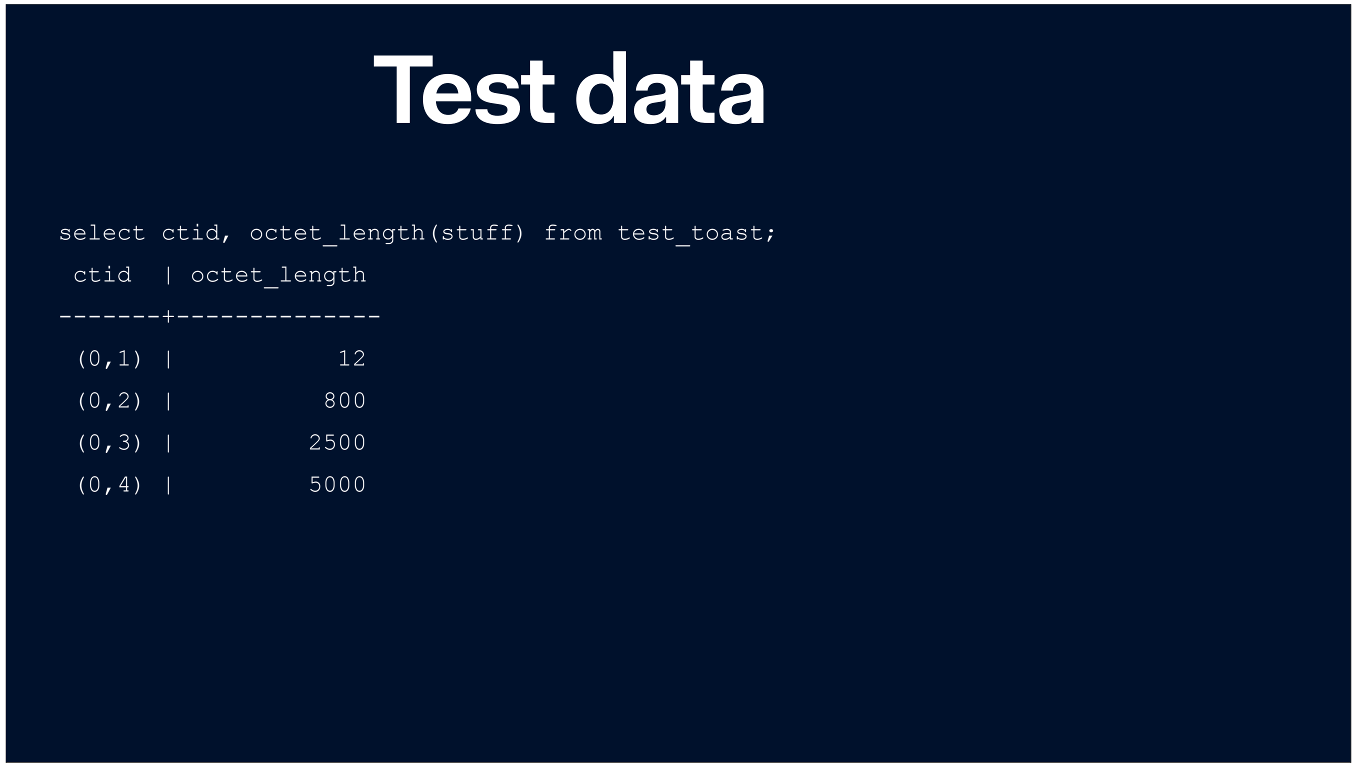
Test data

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



Test data

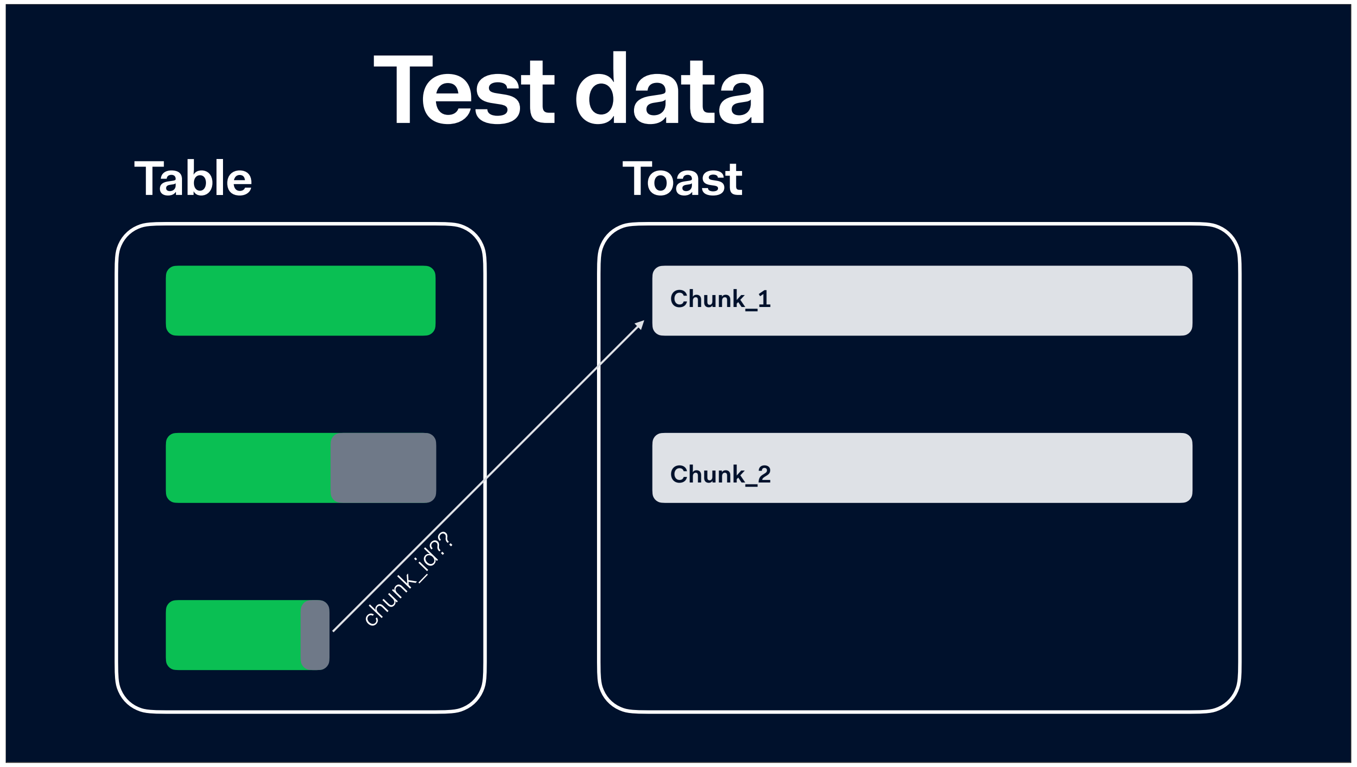
```
create table test_toast(stuff varchar);
insert into test_toast values (’short string');
insert into test_toast (SELECT '800 length string:  ' | <800 - 19  random chars> );
insert into test_toast (SELECT '2500 length string: ' | <2500 - 20 random chars> );
insert into test_toast (SELECT '5000 length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



Test data

```
select ctid, octet_length(stuff) 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.



Test data

Table

Toast

Chunk_1

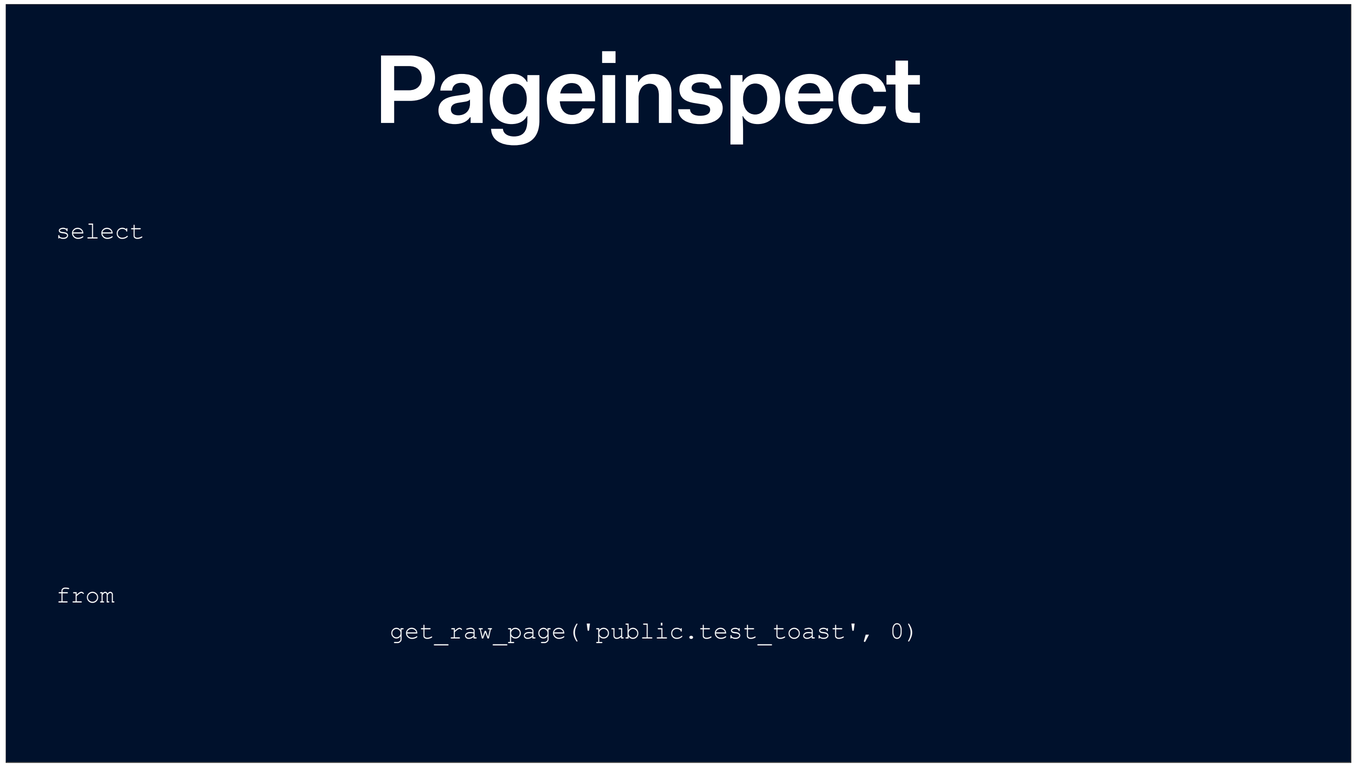
Chunk_2

chunk_id??

How do we find the chunk_id

Pageinspect

How to find toast data?



Pageinspect

```
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

Pageinspect

```
select









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

Pageinspect

```
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 ;
```

Pageinspect

```
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 ;
```

Pageinspect

```
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 ;
```

Pageinspect

```
select


  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')

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

Pageinspect

```
select


  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')

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

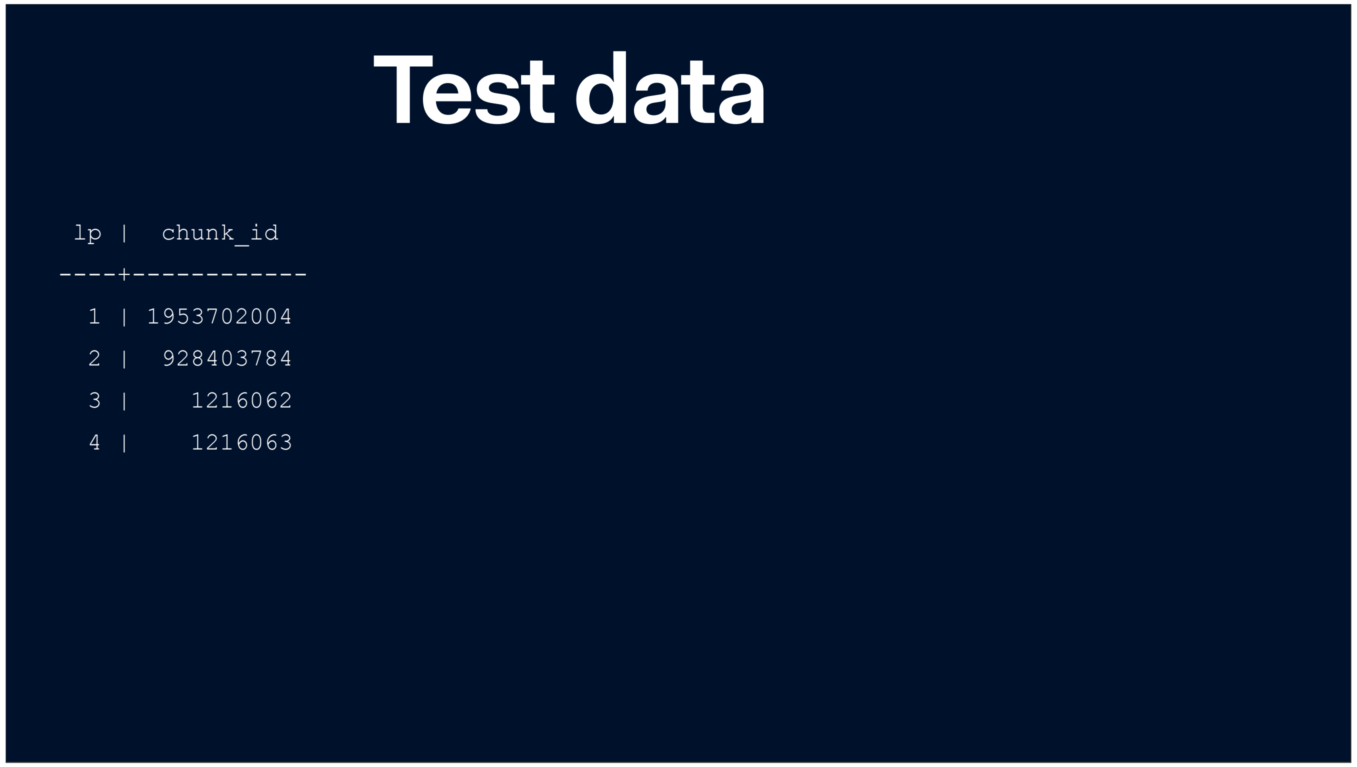
12345678 -> 78563412

Pageinspect

```
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 ;
```

Pageinspect

```
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 ;
```

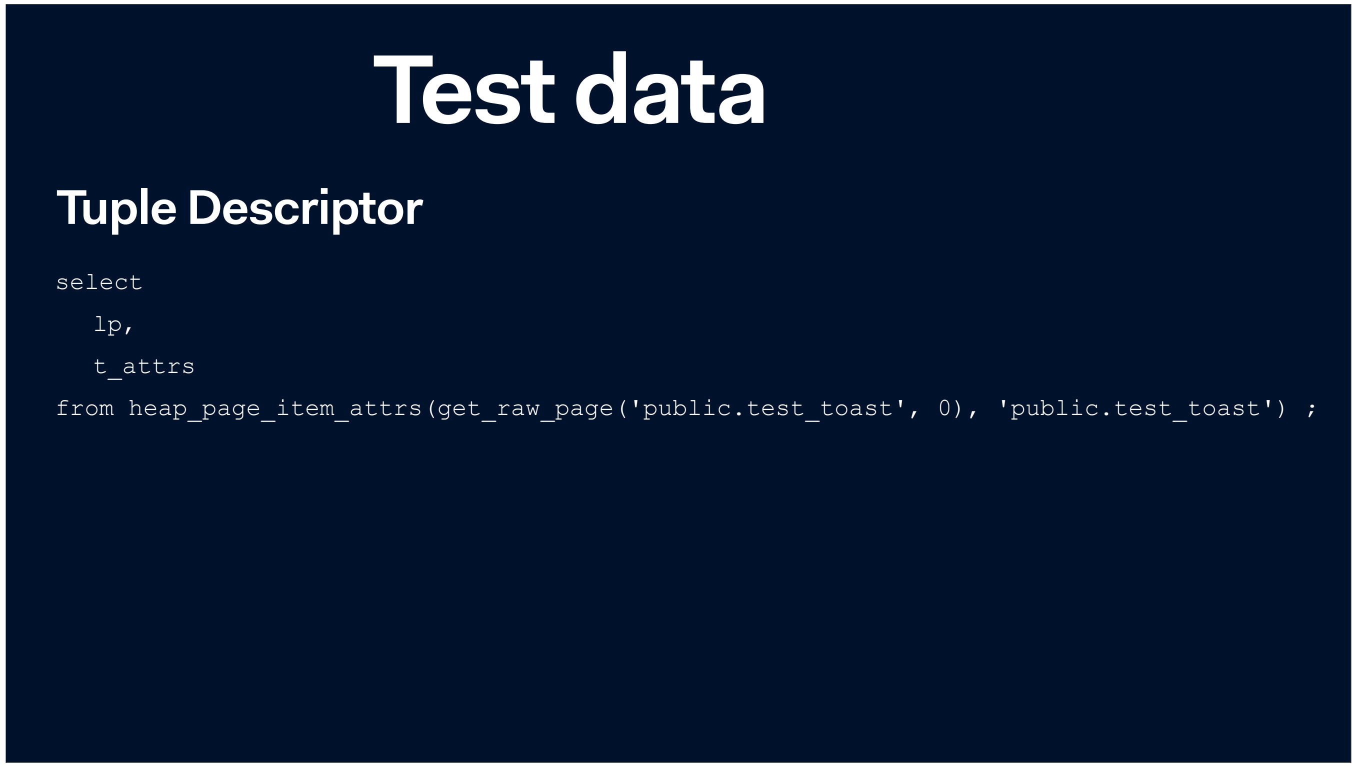


Test data

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

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

Mental break. Make a joke.



Test data

Tuple Descriptor

```
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

Test data

Tuple Descriptor

```
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 | {“\\x900c0000383030206c656e67746820737472696e673a20554830343749373850345….”}
  3 | {"\\x0112c8090000c40900003e8e1200368e1200"}
  4 | {"\\x01128c130000881300003f8e1200368e1200"}
```

Test data

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

Test data

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

 lp | short | header_byte
----+-------+-------------
  1 |     1 |          27
  2 |     0 |         144
  3 |     1 |           1
  4 |     1 |           1
```

Test data

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

 lp | short | header_byte
----+-------+-------------
  1 |     1 |          27
  2 |     0 |         144
  3 |     1 |           1
  4 |     1 |           1
```

Only when short AND header_byte = 1 —> toasted data

Test data

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

 lp | short | header_byte
----+-------+-------------
  1 |     1 |          27
  2 |     0 |         144
  3 |     1 |           1
  4 |     1 |           1
```

Only when short AND header_byte = 1 —> toasted data

Test data

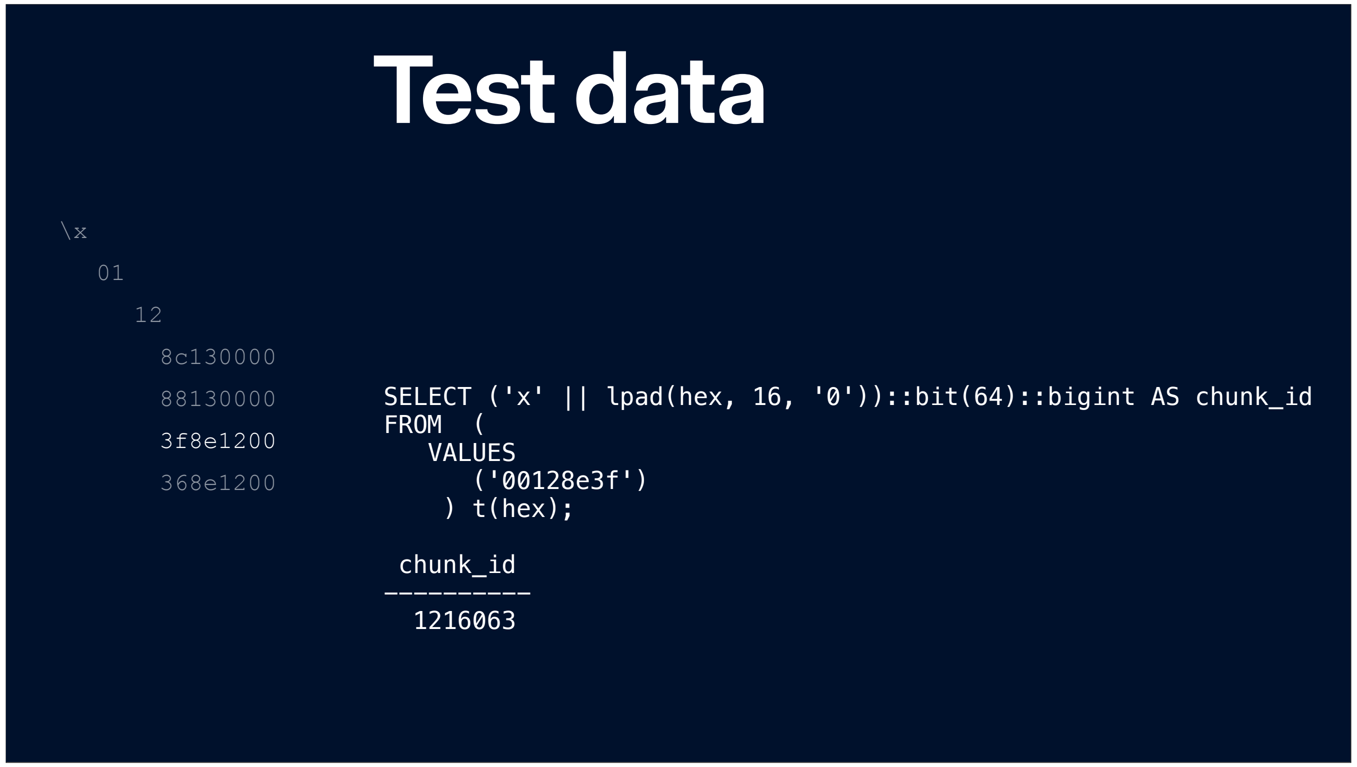
```
select
   lp,
   t_attrs
from heap_page_item_attrs(get_raw_page('public.test_toast', 0), 'public.test_toast') ;

 lp |  t_attrs
——+————————————————————————————————————————————————————————————————————————————————-
  1 | {"\\x1b73686f727420737472696e67"}
  2 | {"\\x900c0000383030206c656e67746820737472696e673a2055483034374937385034 5….”}
  3 | {"\\x0112c8090000c40900003e8e1200368e1200"}
  4 | {"\\x01128c1300008813000 03f8e1200368e1200"}
```

Test data

```
\x01128c130000881300003f8e1200368e1200
```

Test data

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

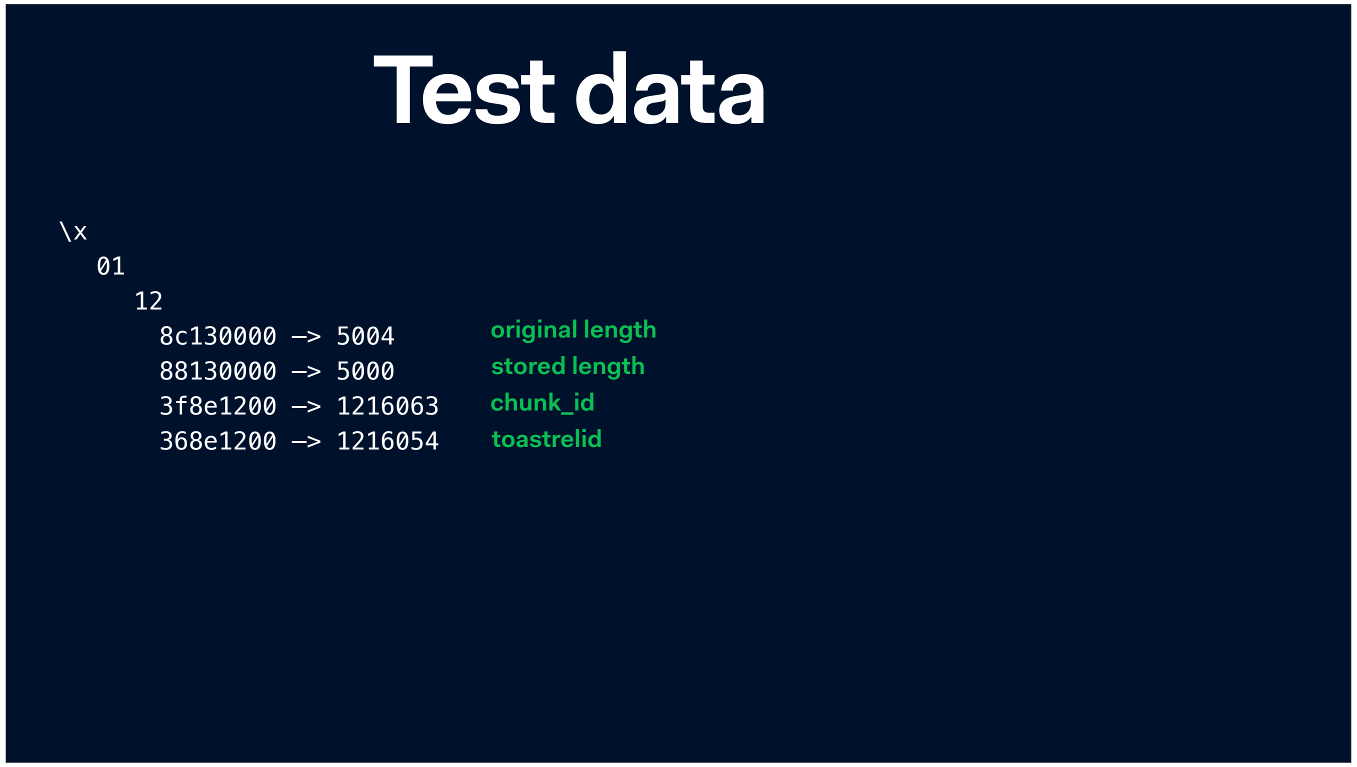


Test data

```
\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
```

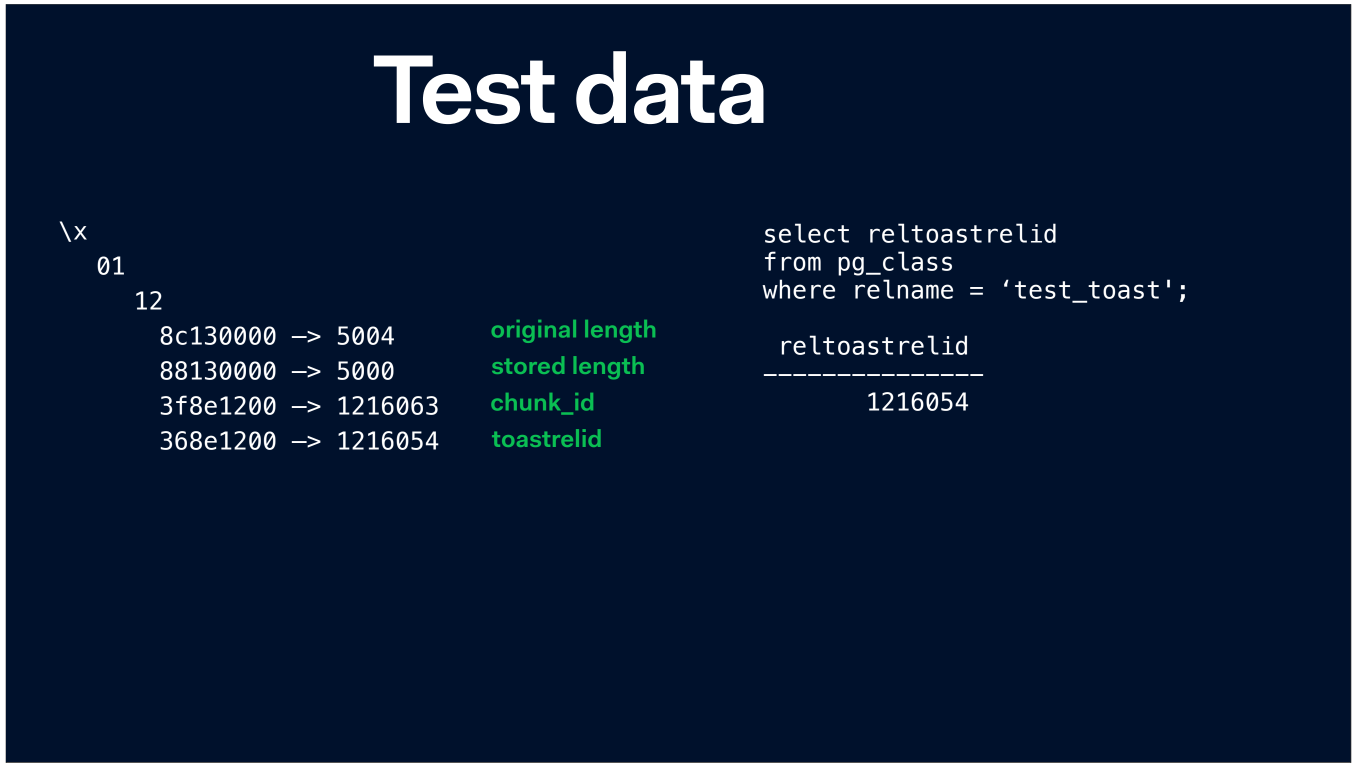
63 * 16^0 + 142 * 16^2 + 18 * 16^4 + 0 * 16^6 = 1216063



Test data

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

Difference between 5004 and 5000 is the 4 byte header.



Test data

```
\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
```

Test data

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

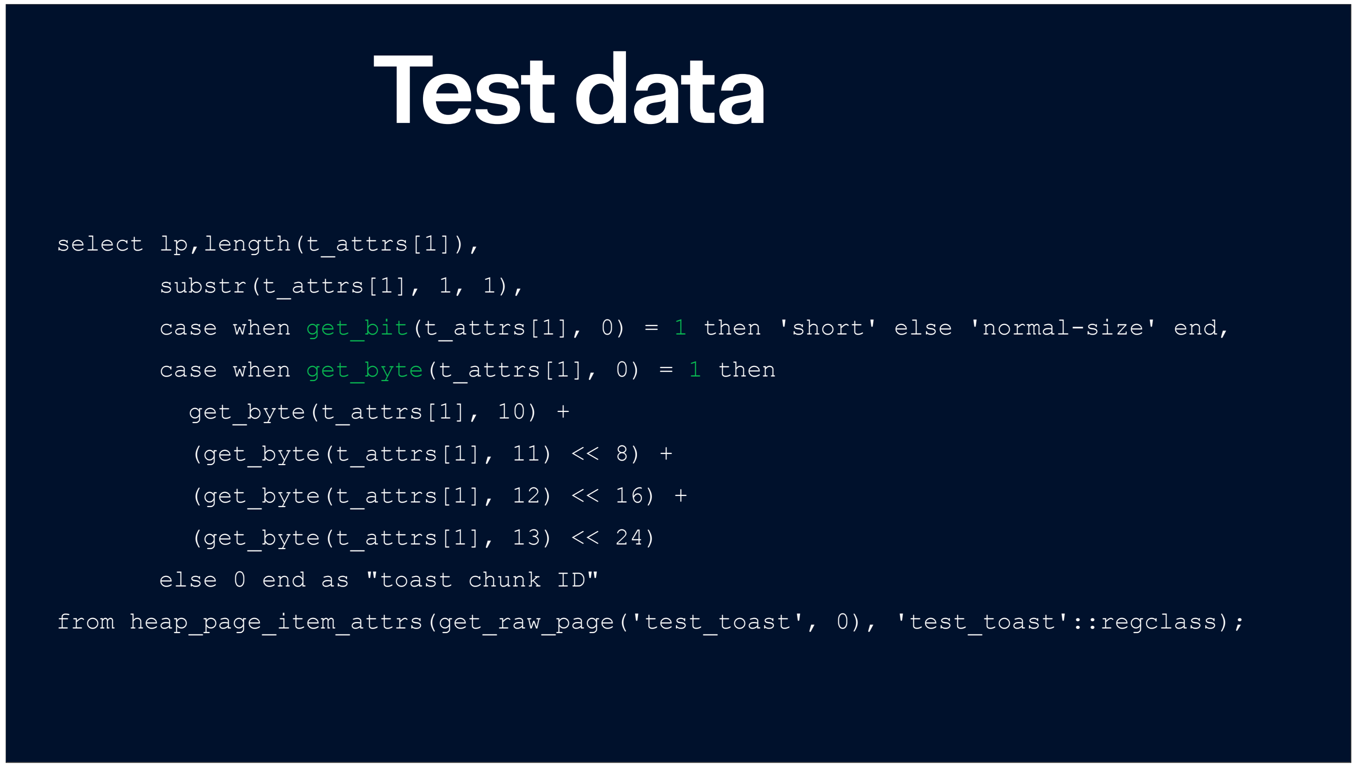
```
select reltoastrelid
from pg_class
where relname = ‘test_toast';

 reltoastrelid
---------------
       1216054
```

```
select 1216054::regclass::text;

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

Test data

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

```
select
  chunk_id, chunk_seq, sum(octet_length(chunk_data))
from pg_toast.pg_toast_1216051
where chunk_id = 1216063
group by grouping sets
  (chunk_id, chunk_seq),()
order by chunk_seq nulls last;

 chunk_id | chunk_seq | sum
----------+-----------+------
          |         0 | 1996
          |         1 | 1996
          |         2 | 1008
  1216063 |           | 5000
```



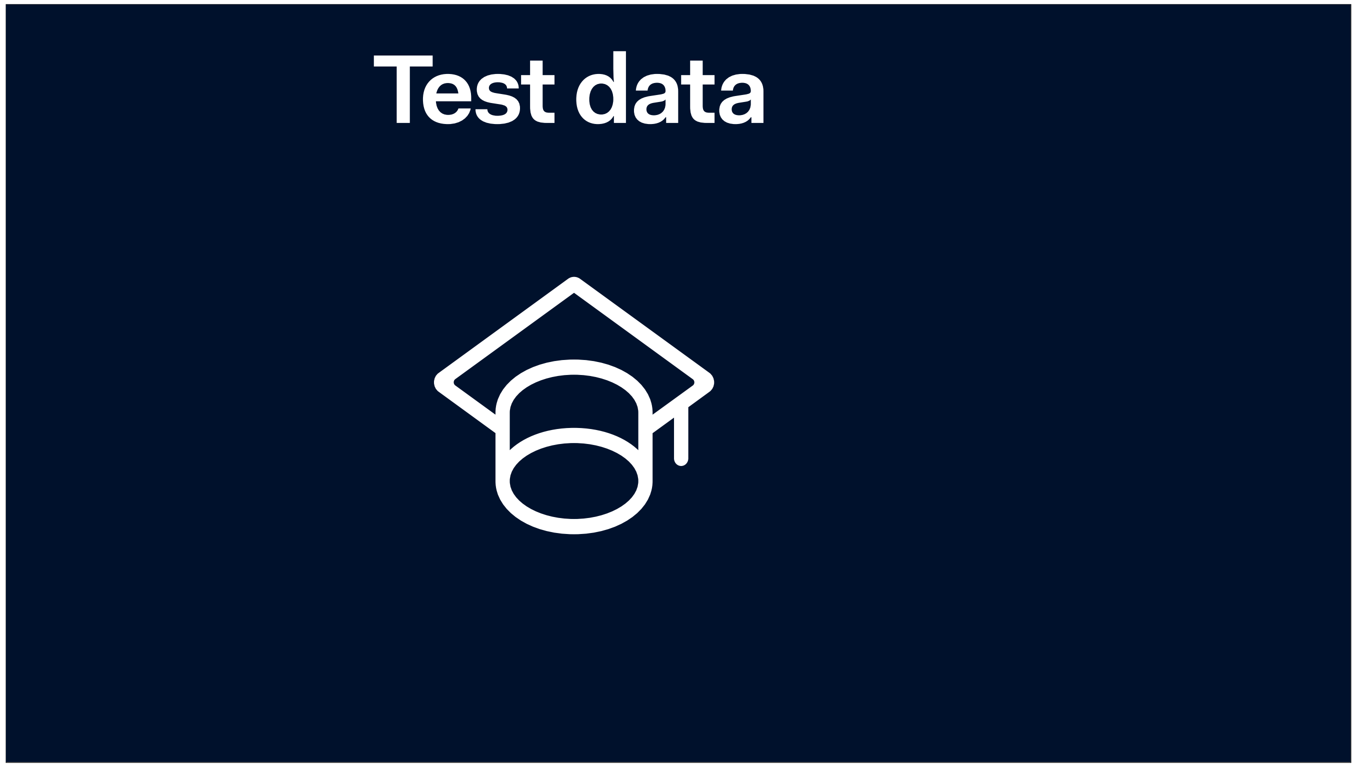
Test data

```
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.

Test data

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



Test data

What did we learn this far

The only thing to understand at this point is that we are now able to read the chunk_id from the main table.

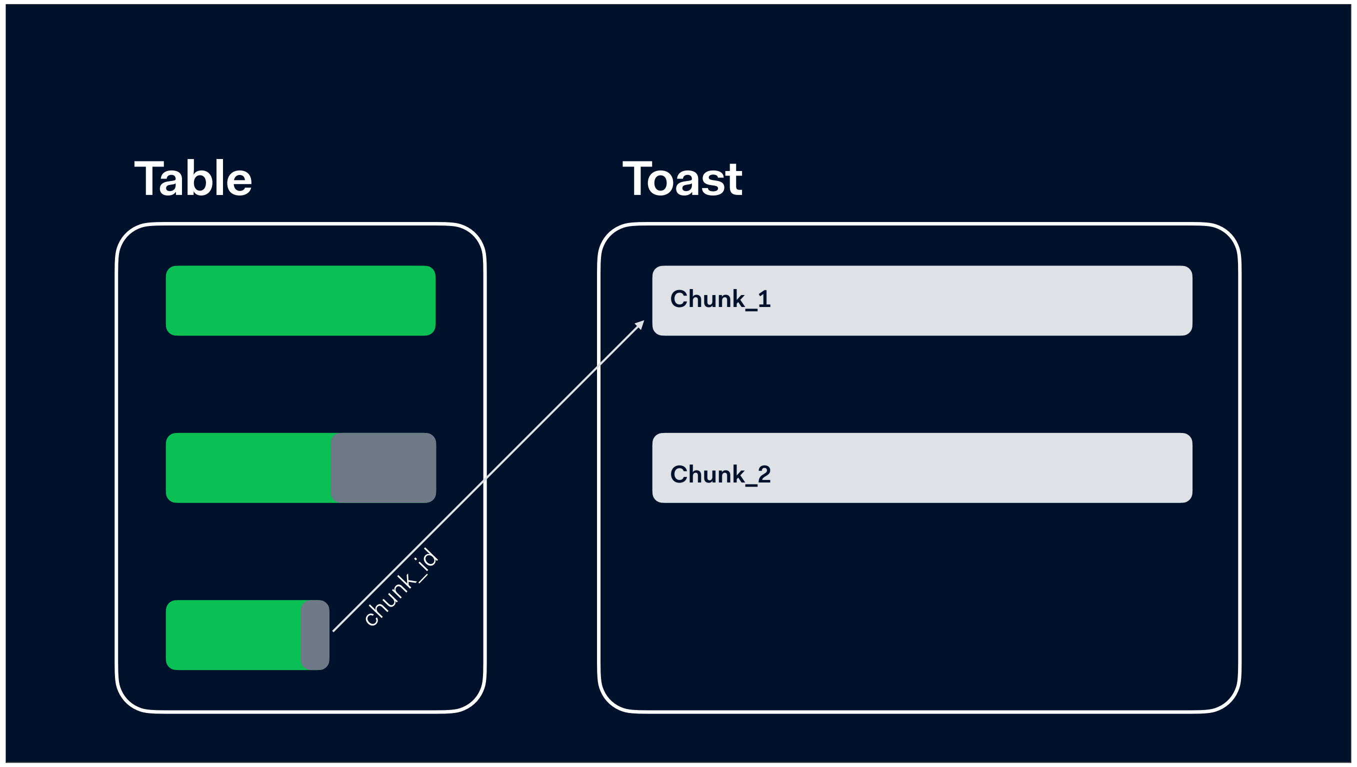
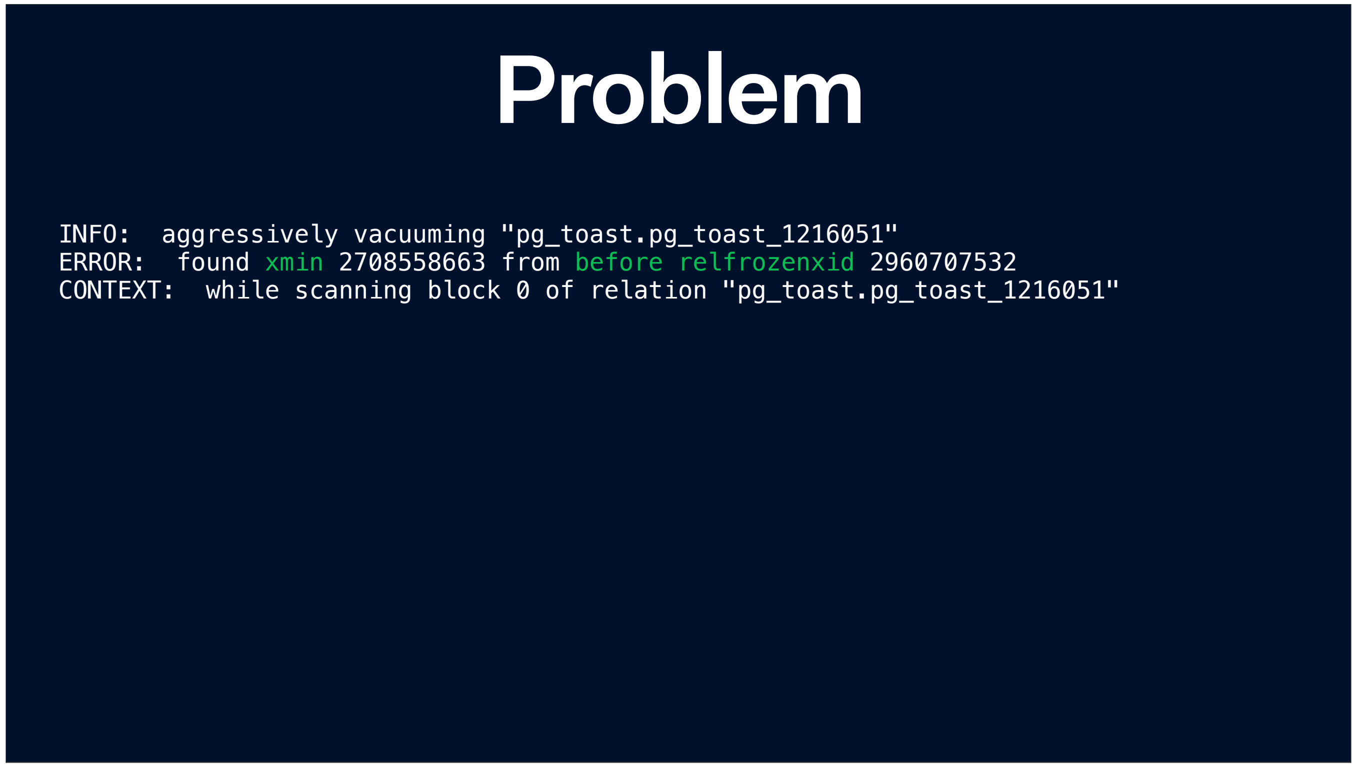


Table
Toast
Chunk_1
Chunk_2
chunk_id

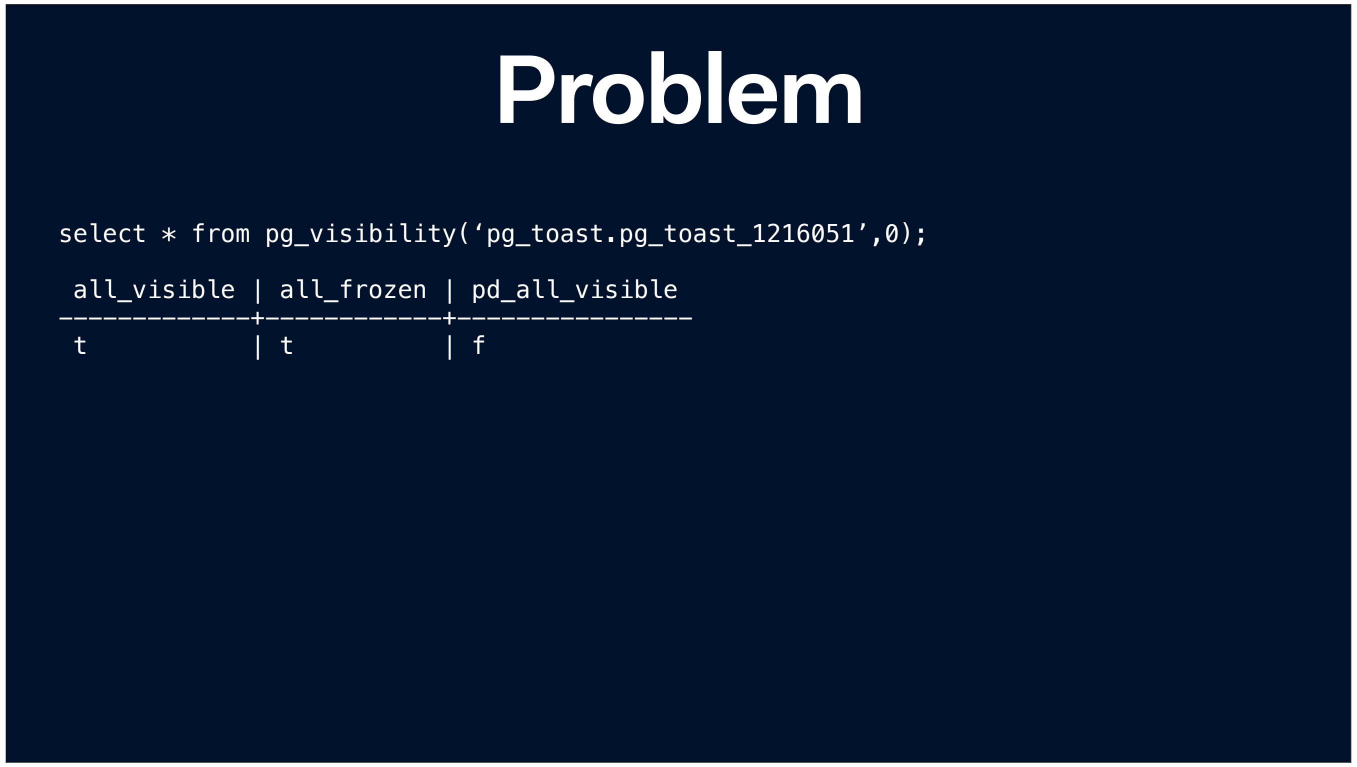
Problem



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

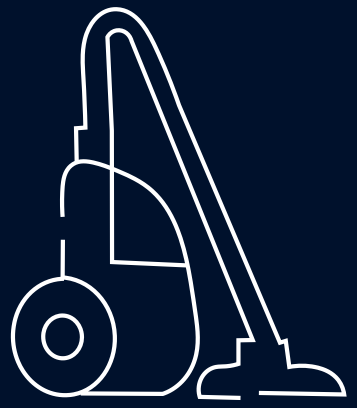
All visible: no vacuum required

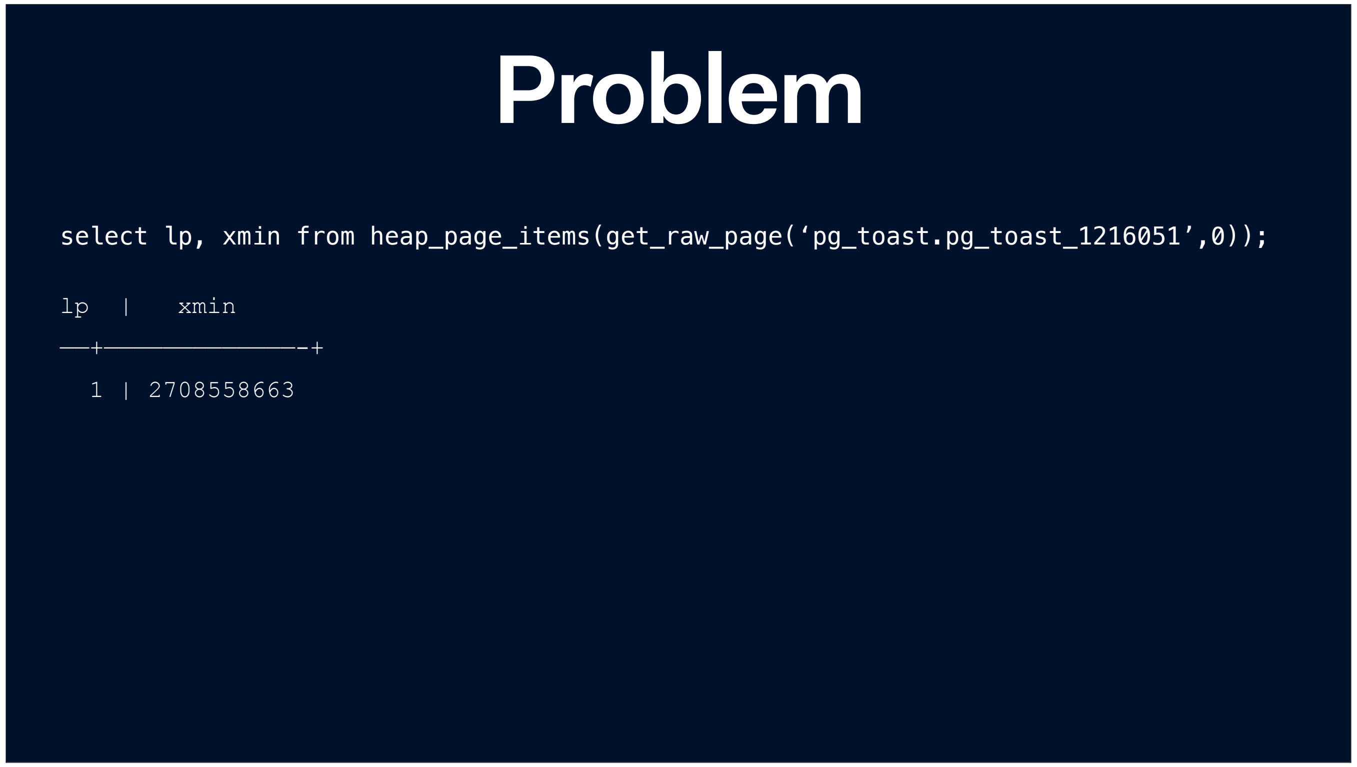
All frozen: all rows are frozen: anti-wraparound vacuum need not revisit the page.

pd_all_frozen: all frozen bit from the page instead as from the visibility map.

Visibility map

0	0
1	1
2	0
3	0
4	0
5	0
6	0
7	1
8	1
9	1



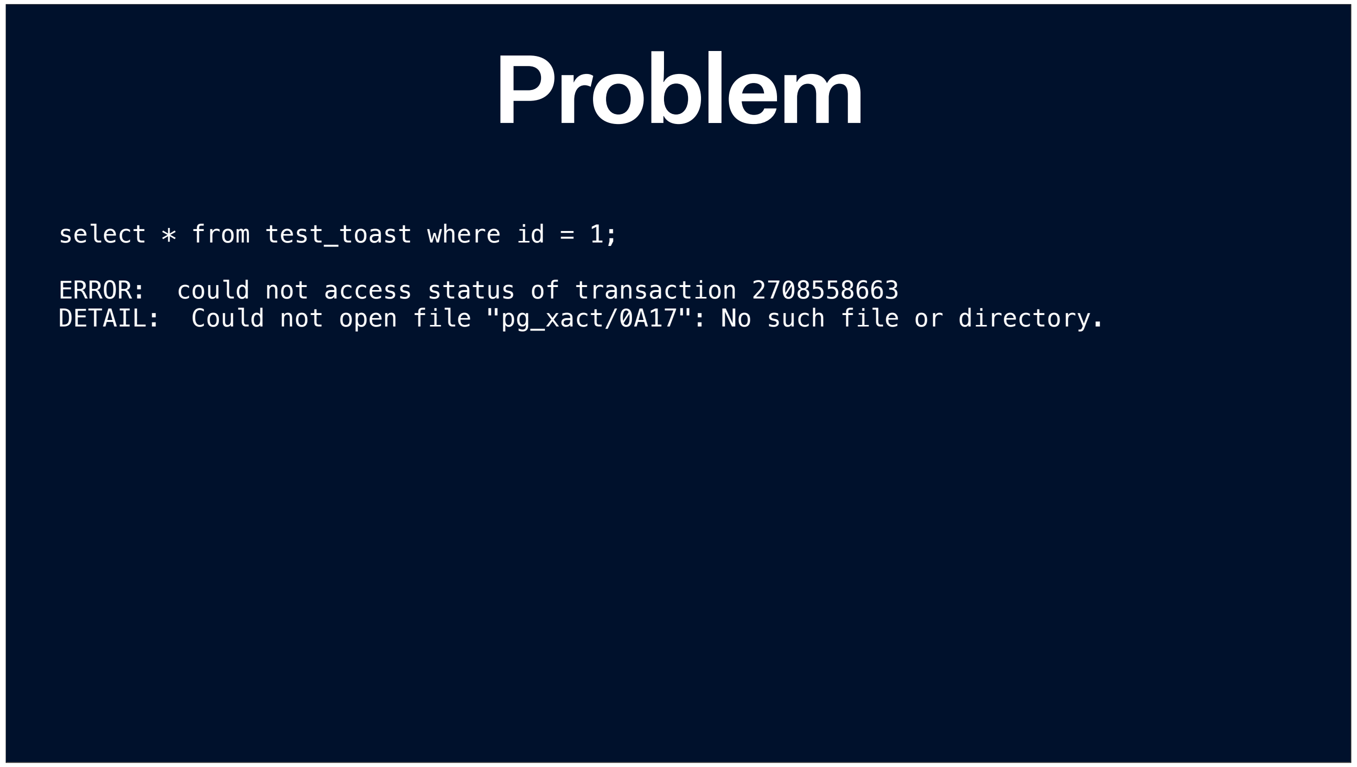


Problem

```
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.

We can't query this directly. We would get an error because of the invalid page. Page inspect to the rescue.

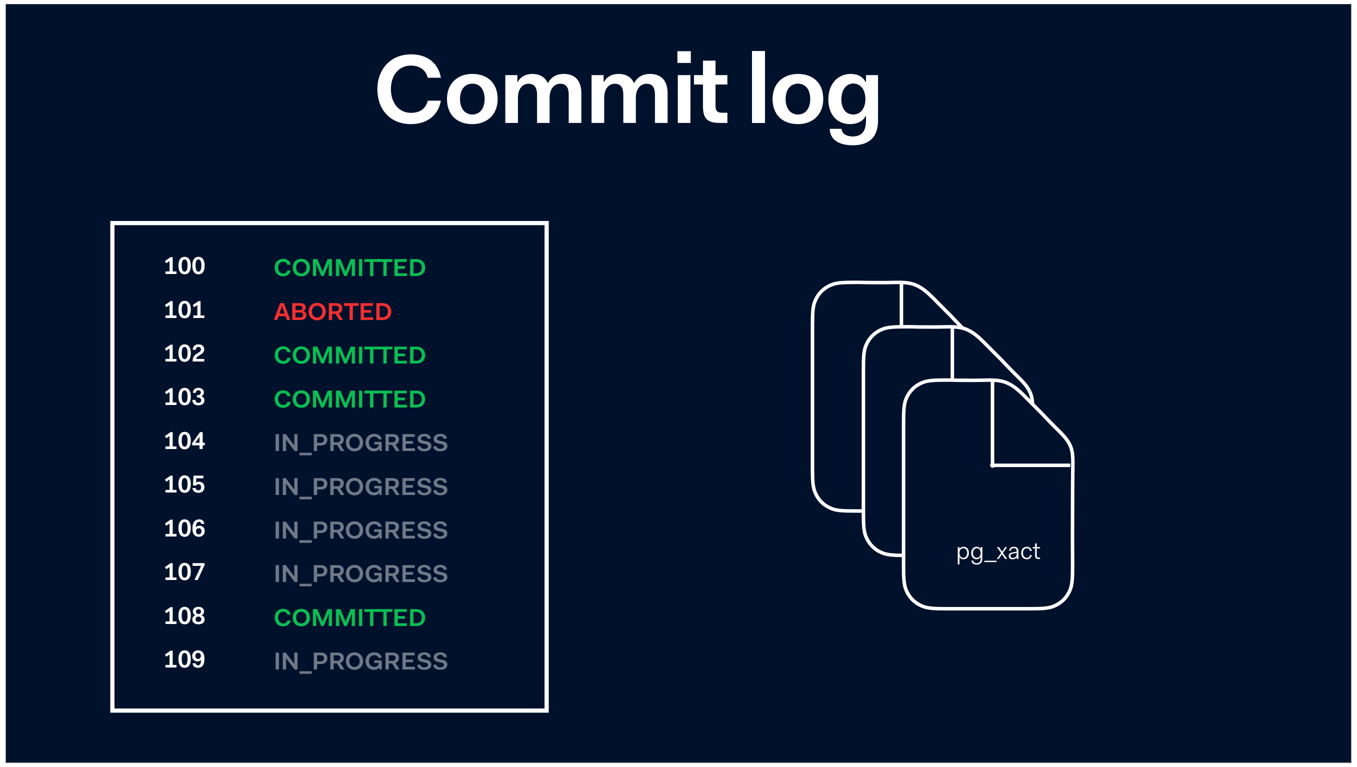


Problem

```
select * from test_toast where id = 1;

ERROR:  could not access status of transaction 2708558663
DETAIL:  Could not open file "pg_xact/0A17": No such file or directory.
```

Try to select all data from the page.

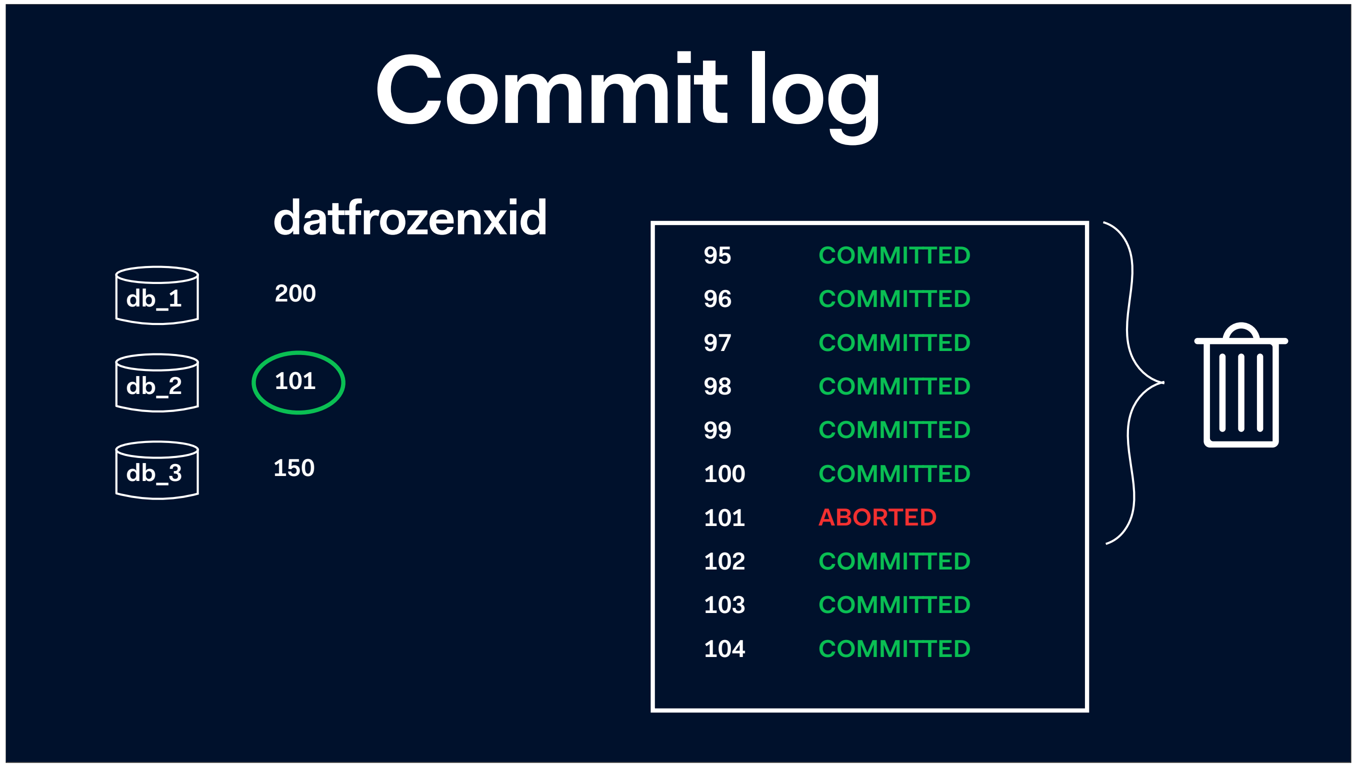
The file pg_xact/0A17 depends on the transaction number and some logic.



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

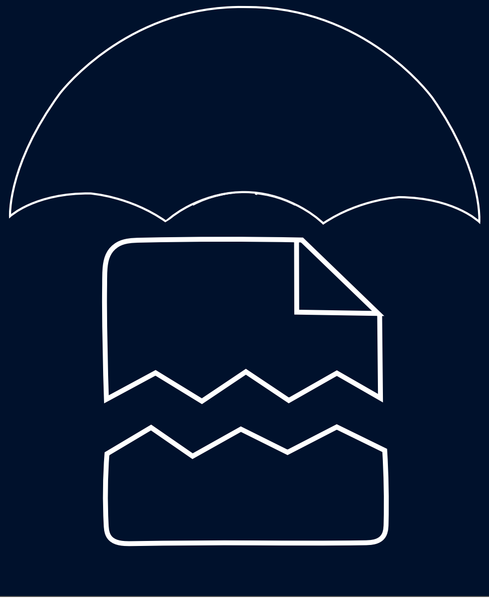


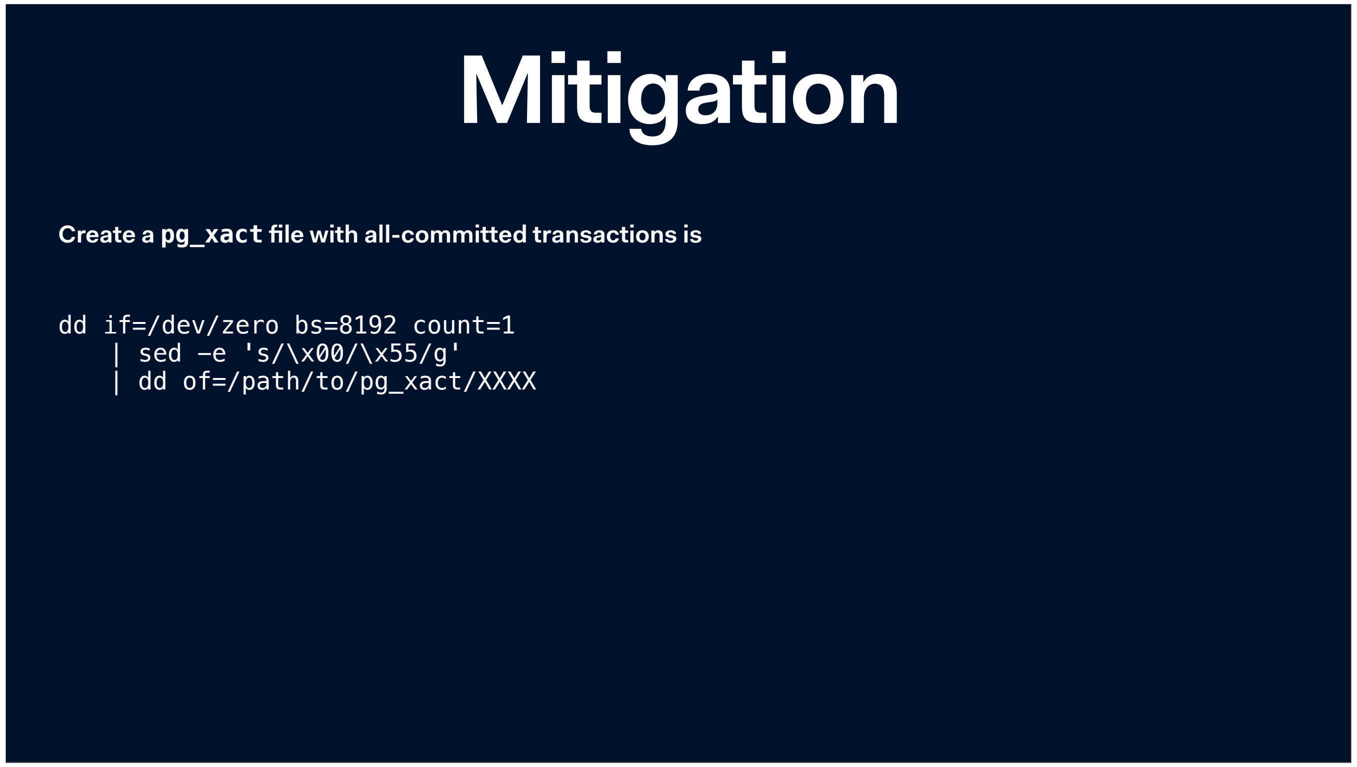
Commit log

	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





Mitigation

Create a `pg_xact` file with all-committed transactions is

```
dd if=/dev/zero bs=8192 count=1
   | sed -e 's/\x00/\x55/g'
   | dd of=/path/to/pg_xact/XXXX
```

Try to create a clog with all transactions marked as being committed.
Didn't work, page is still a mess.
Fortunately, it was a bad plan anyway.

Mitigation

pg_surgery

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

Mitigation

pg_surgery

Mitigation

pg_surgery

`heap_force_kill`

Mitigation

pg_surgery

```
heap_force_kill
heap_force_freeze
```

Mitigation

```
create extension pg_surgery ;
```

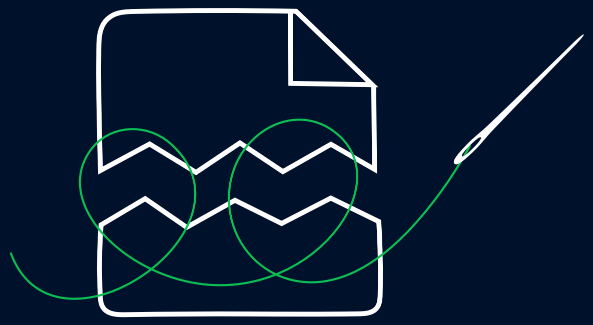
Mitigation

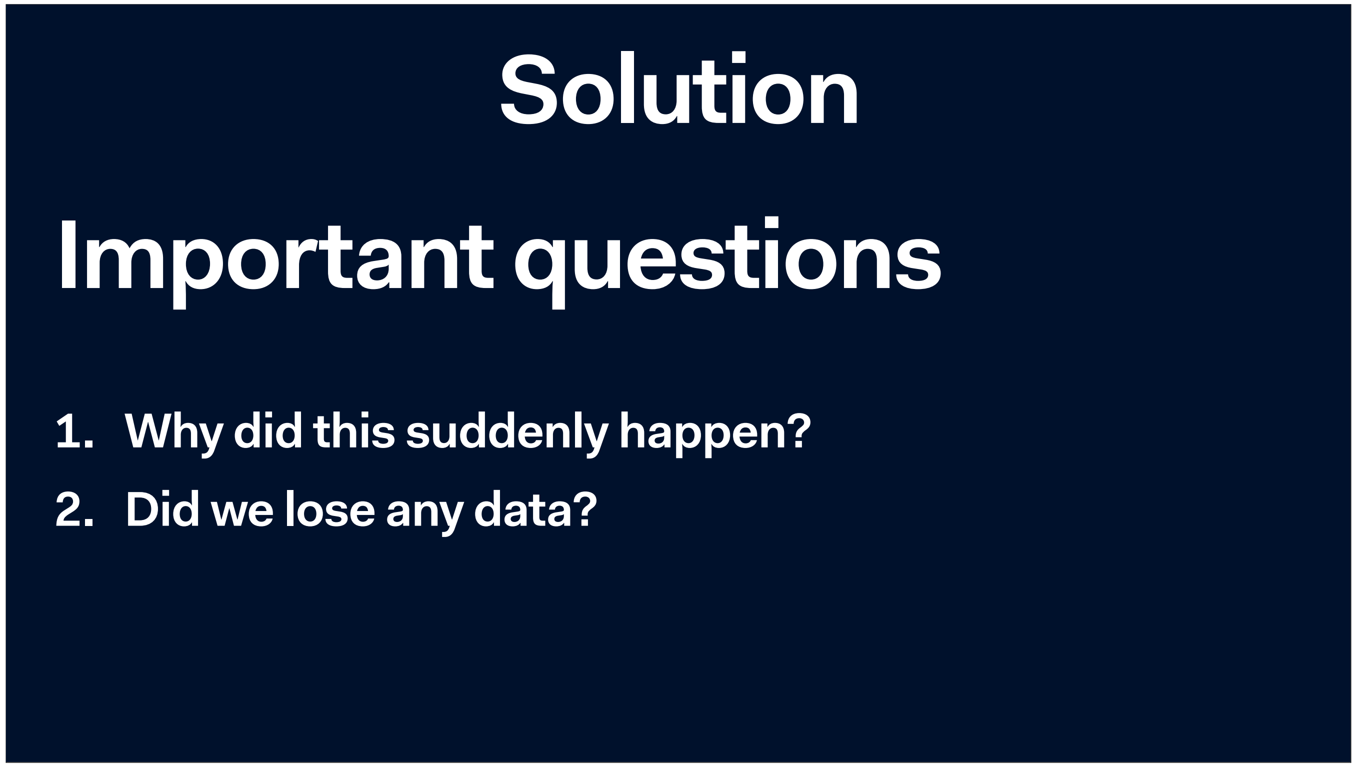
```
create extension pg_surgery ;


SELECT heap_force_freeze('pg_toast.pg_toast_1216051', ‘{"(0,1)"}');

 heap_force_freeze
-------------------

(1 row)
```

Mitigation

```
create extension pg_surgery ;

SELECT heap_force_freeze('pg_toast.pg_toast_1216051', ‘{"(0,1)"}');

 heap_force_freeze
-------------------

(1 row)

select * from test_toast where id = 1;
   id  |        stuff
-------+-------------
     1 | short string
```

Solution



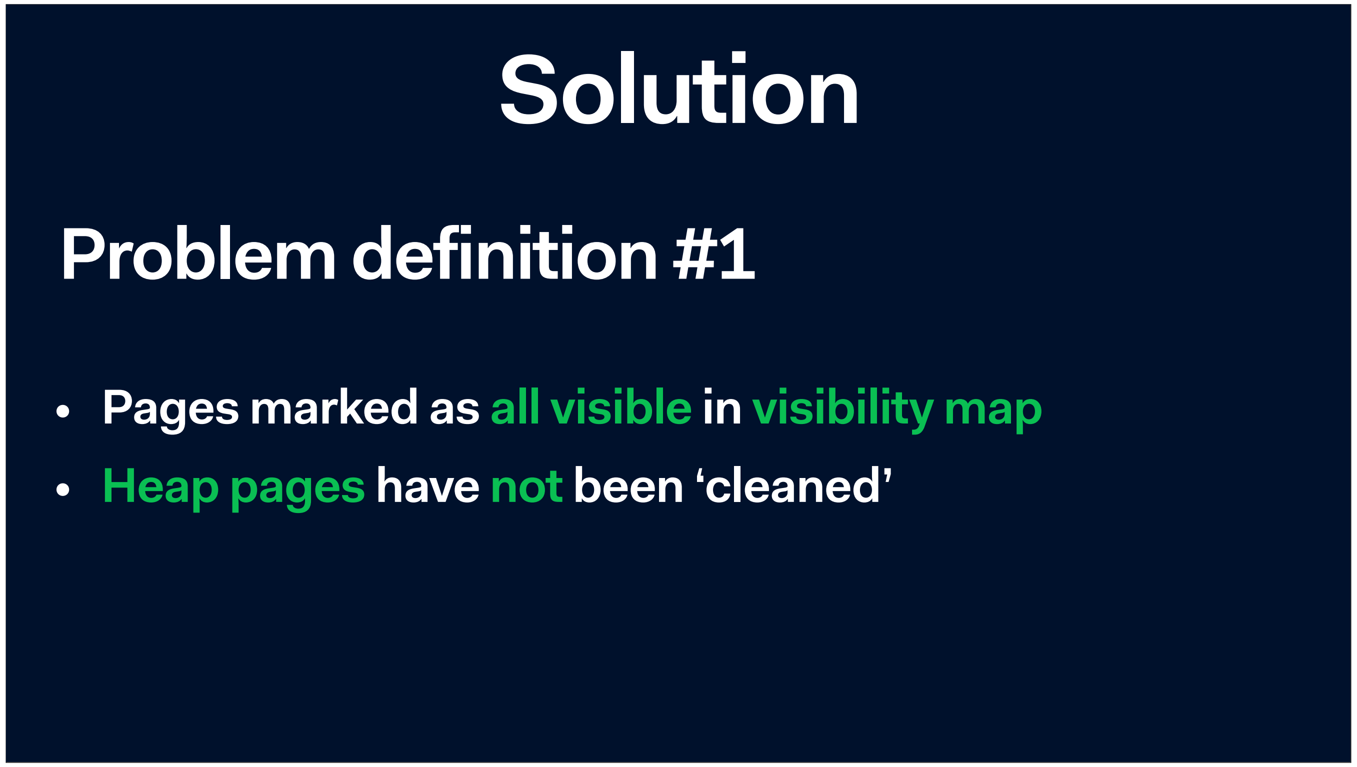


Solution

Important questions

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

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



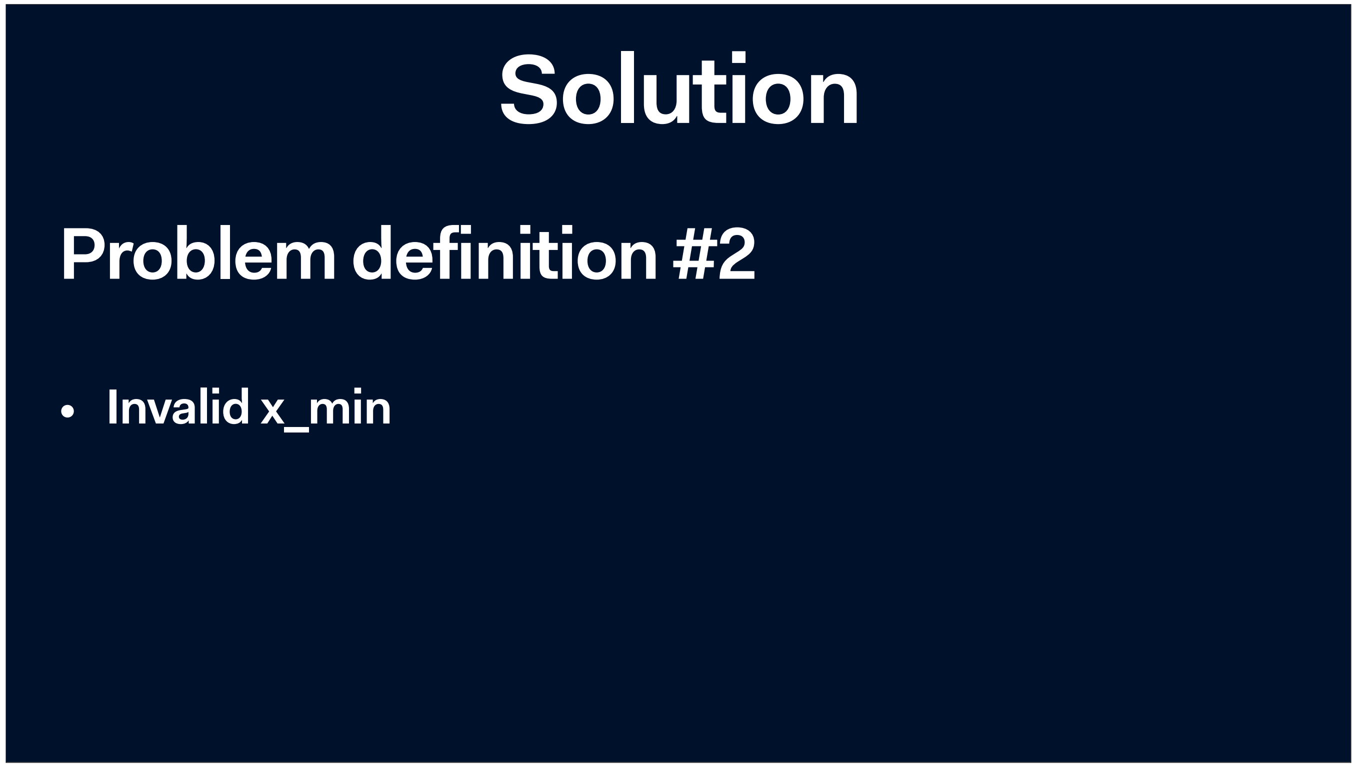
Solution

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 cleaned up by vacuum or single page cleanup.



Solution

Problem definition #2

- Invalid x_min

Solution

Problem definition #2

- Invalid x_min
- Invalid x_max

Solution

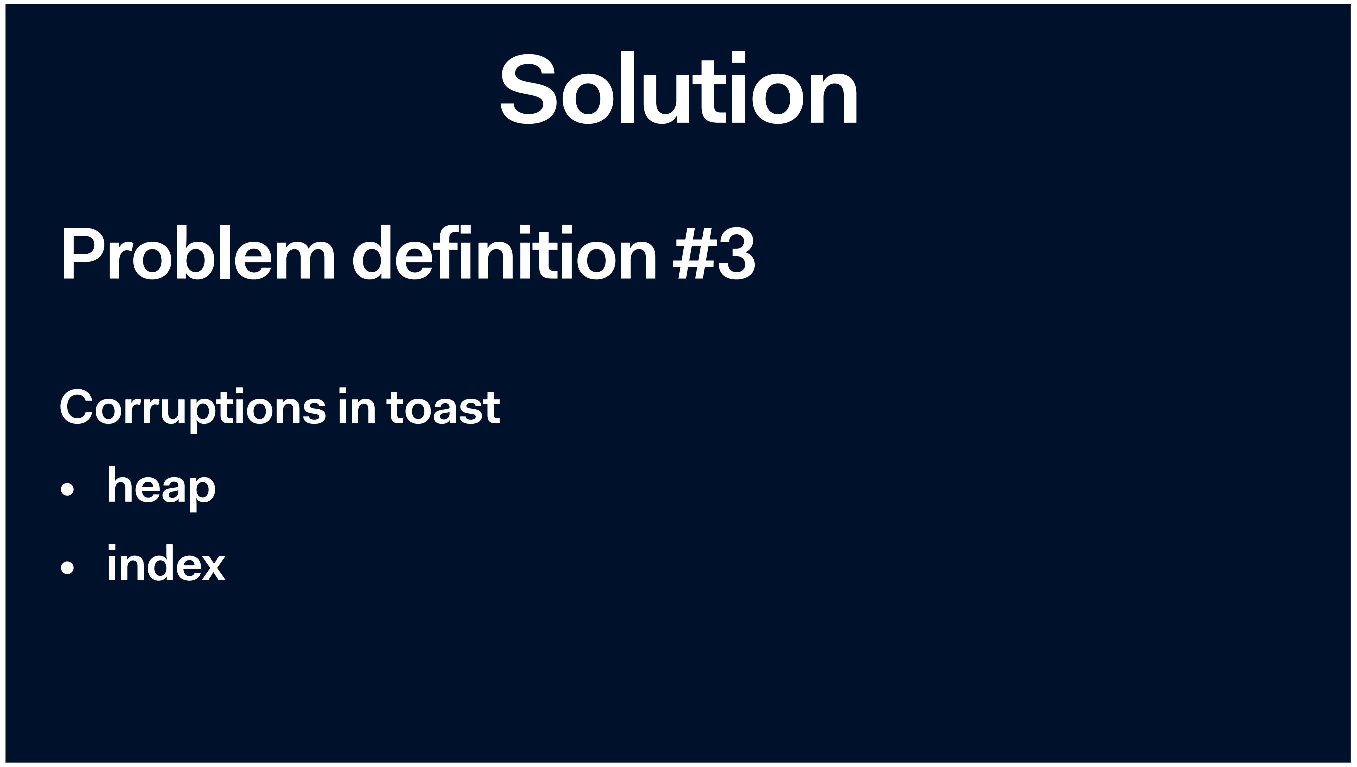
Problem definition #2

- Invalid x_min
- Invalid x_max
- missing chunk number 0 for toast value

Solution

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

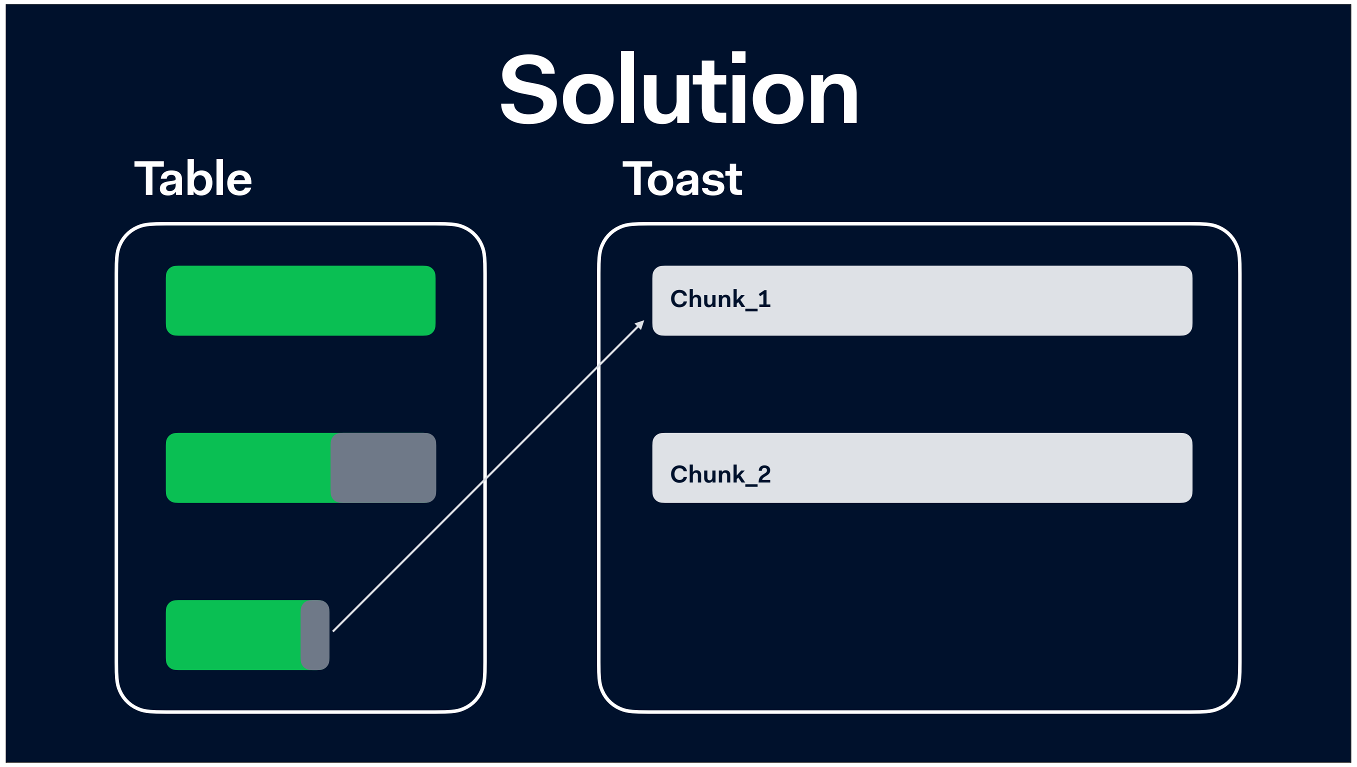


Solution

Problem definition #3

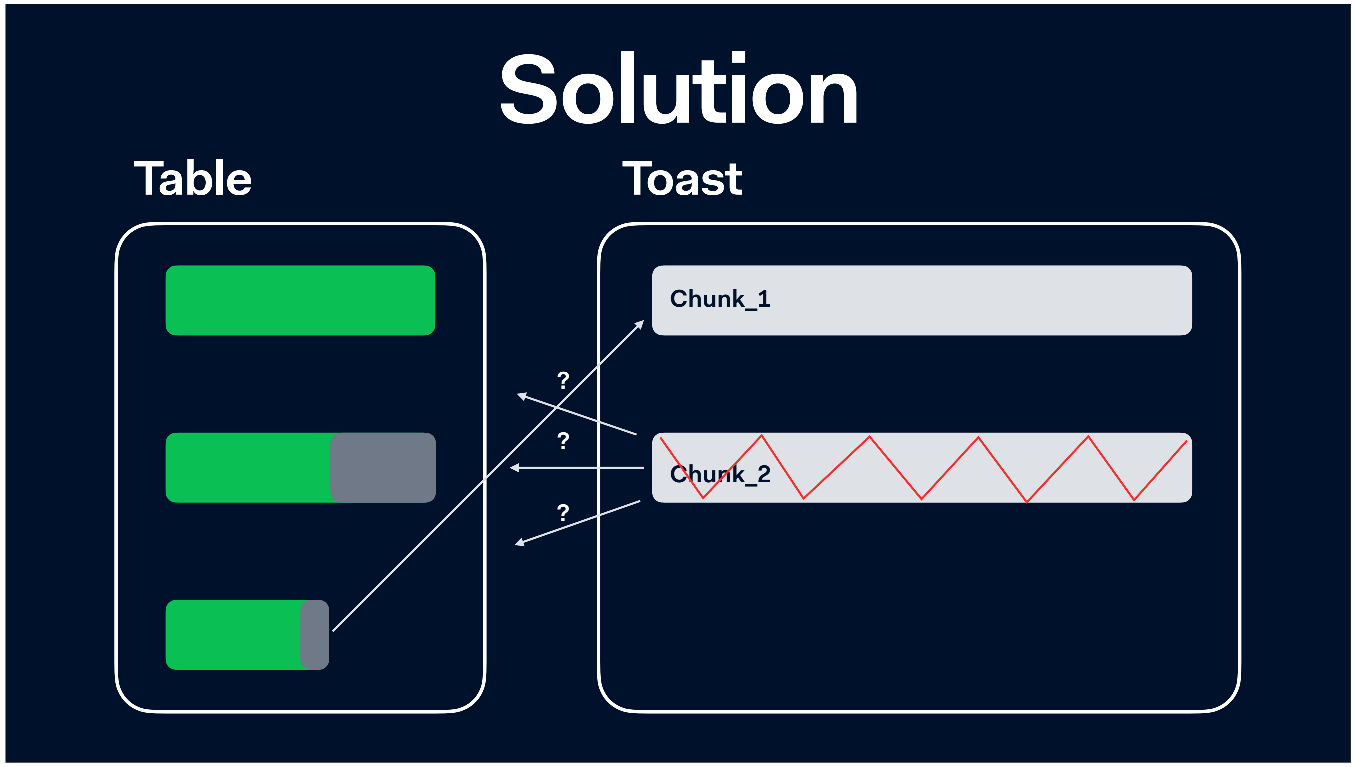
Corruptions in toast

- heap
- index



Solution

From table to toast is easy.



Solution

Table

Toast

Chunk_1

Chunk_2

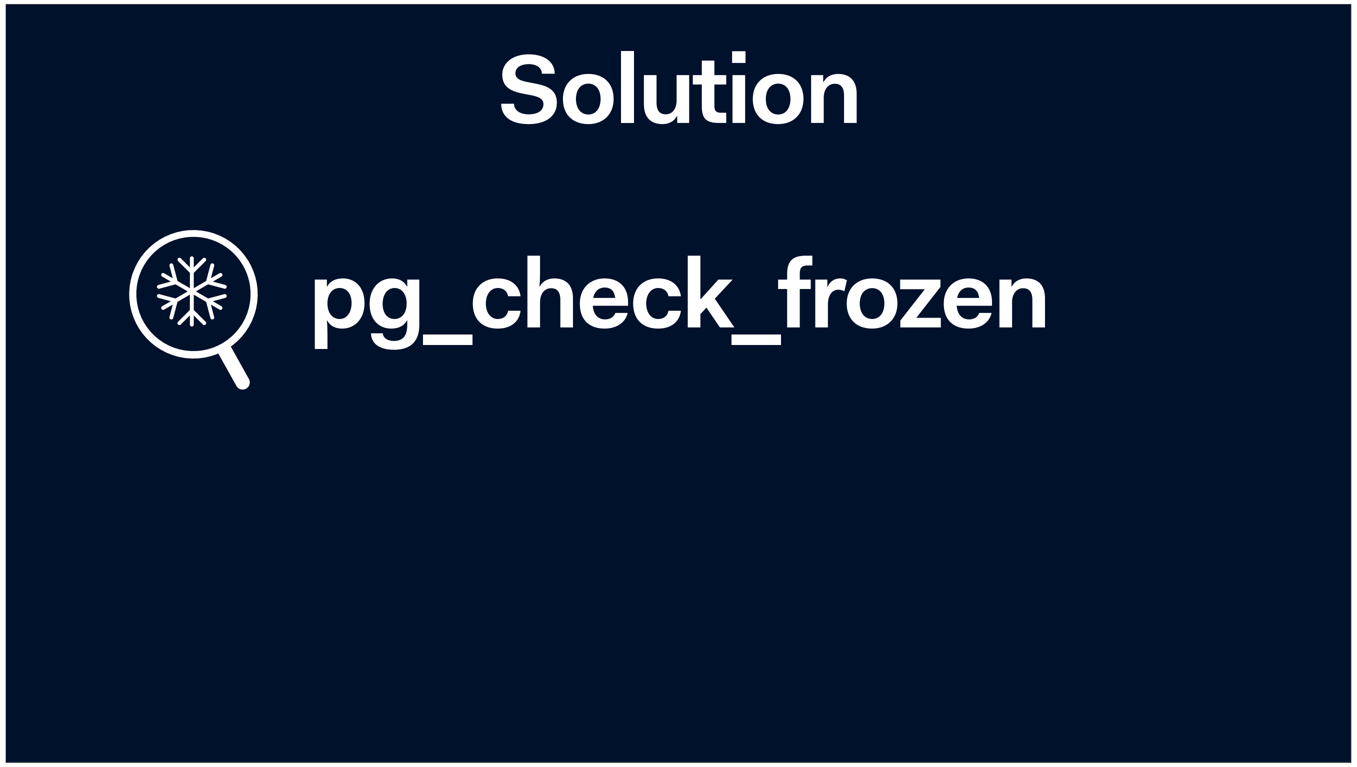
?

?

?

From toast to table back is much harder.

What we need is a map

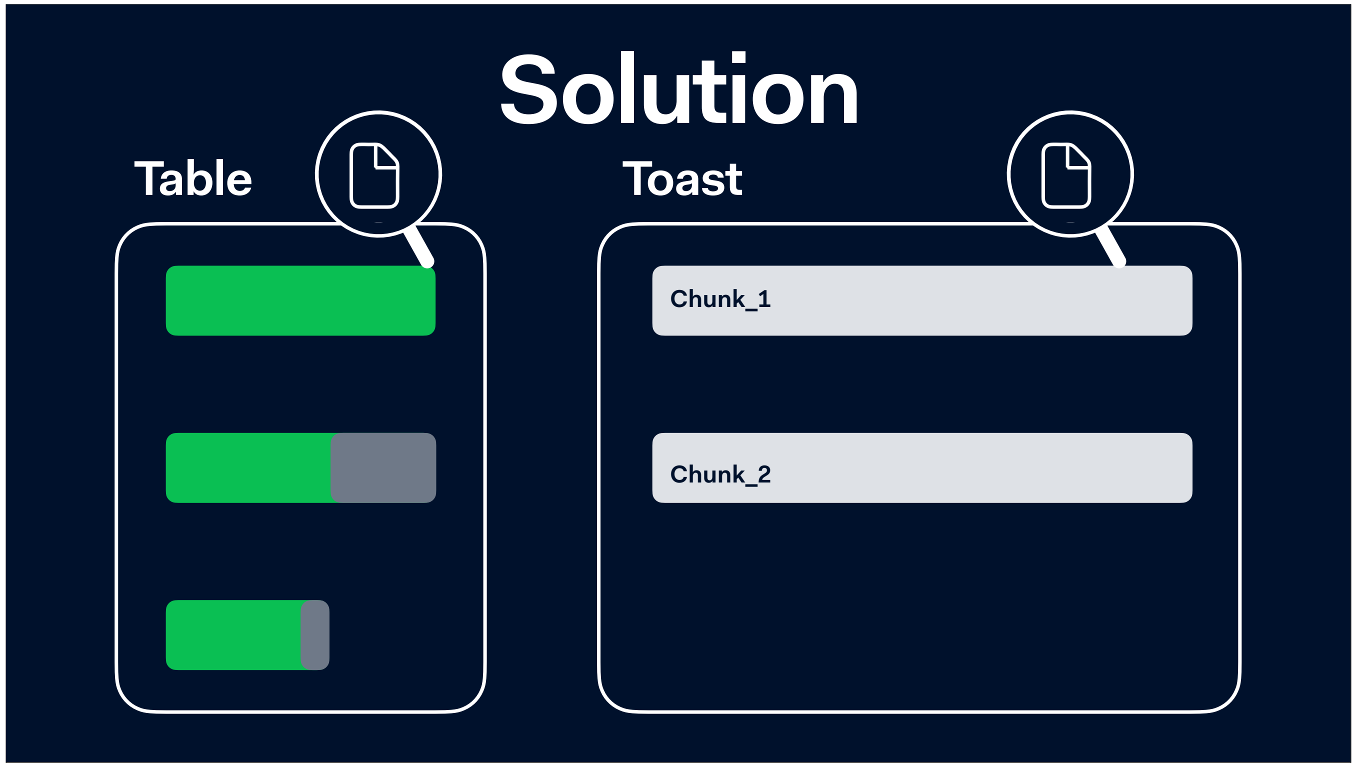


Solution

pg_check_frozen

Run pg_check_frozen from the pg_visibility extension

Our problem is within a limited range.
The only reason I took the image of Chicago is the location of this conference.



Solution

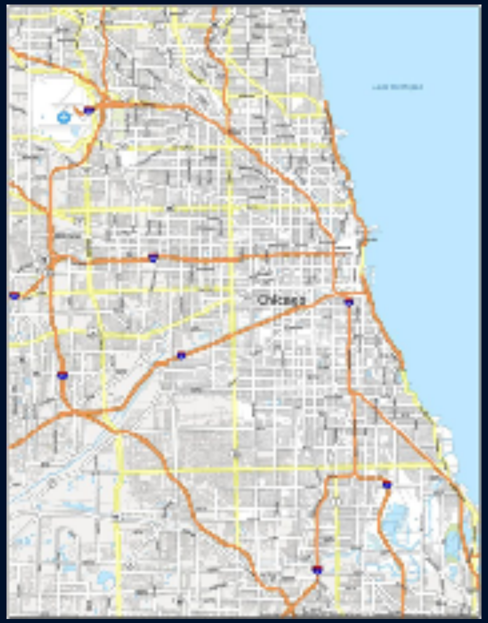
Table

Toast

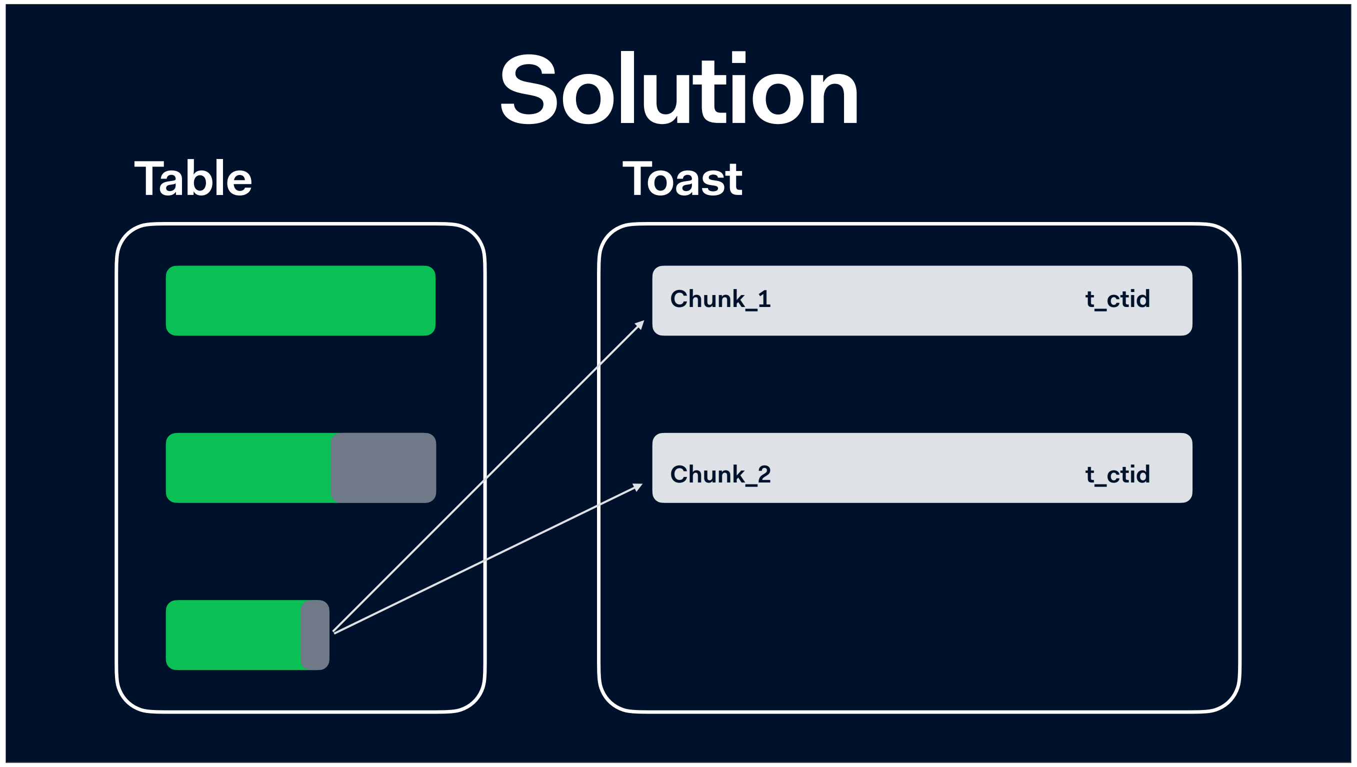
Chunk_1

Chunk_2

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.

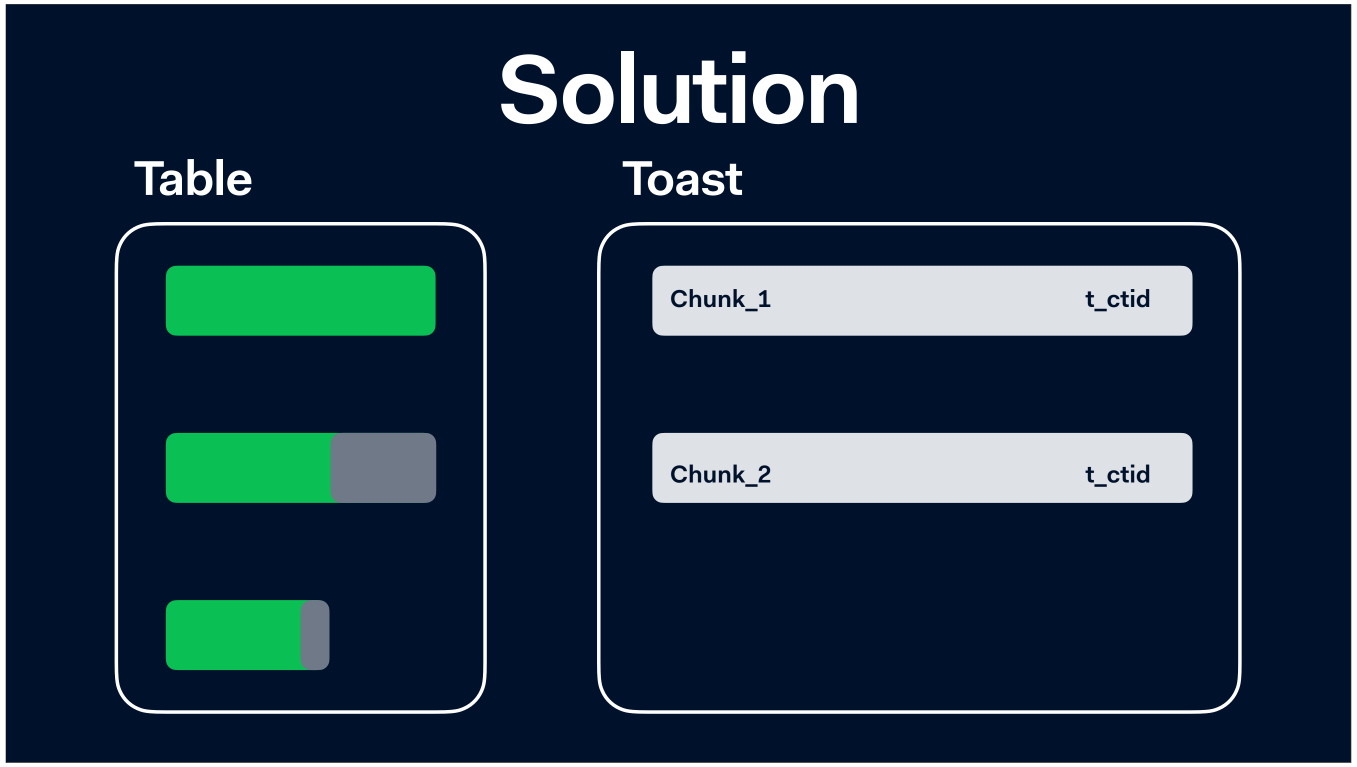


Now we have a map.



Solution

From table to toast is easy.



Solution

Table

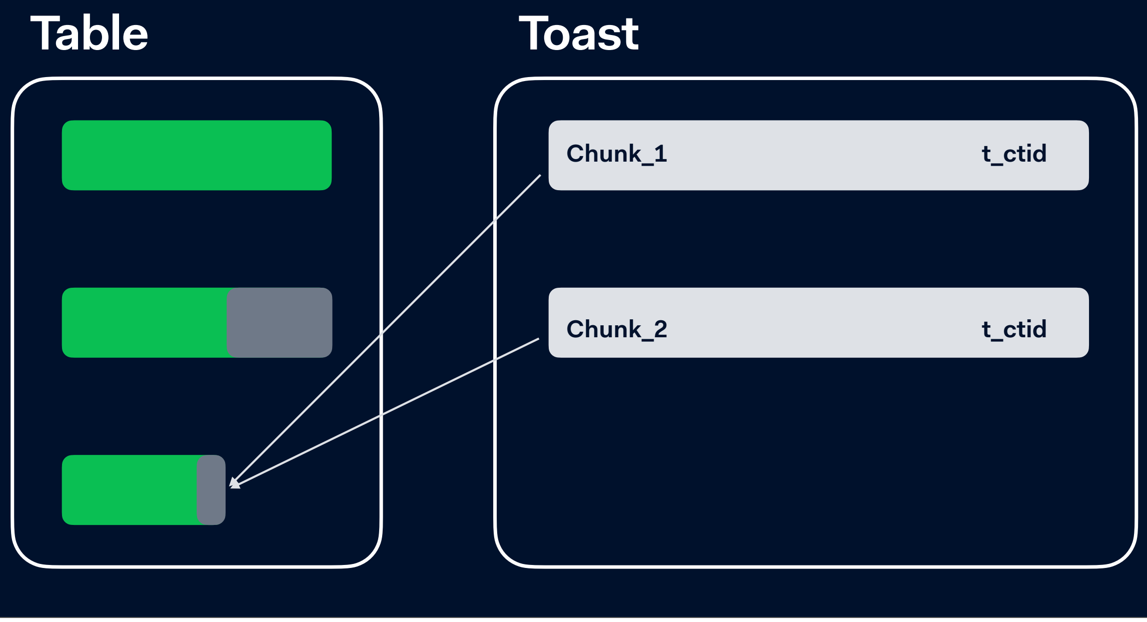
Toast

Chunk_1 t_ctid

Chunk_2 t_ctid

With the map we can go from toast to main

Solution



Table

Toast

Chunk_1 t_ctid

Chunk_2 t_ctid

Solution

No data loss

Solution

No data loss

All corruptions originate from the two days before the upgrade

Solution

Toast data exists

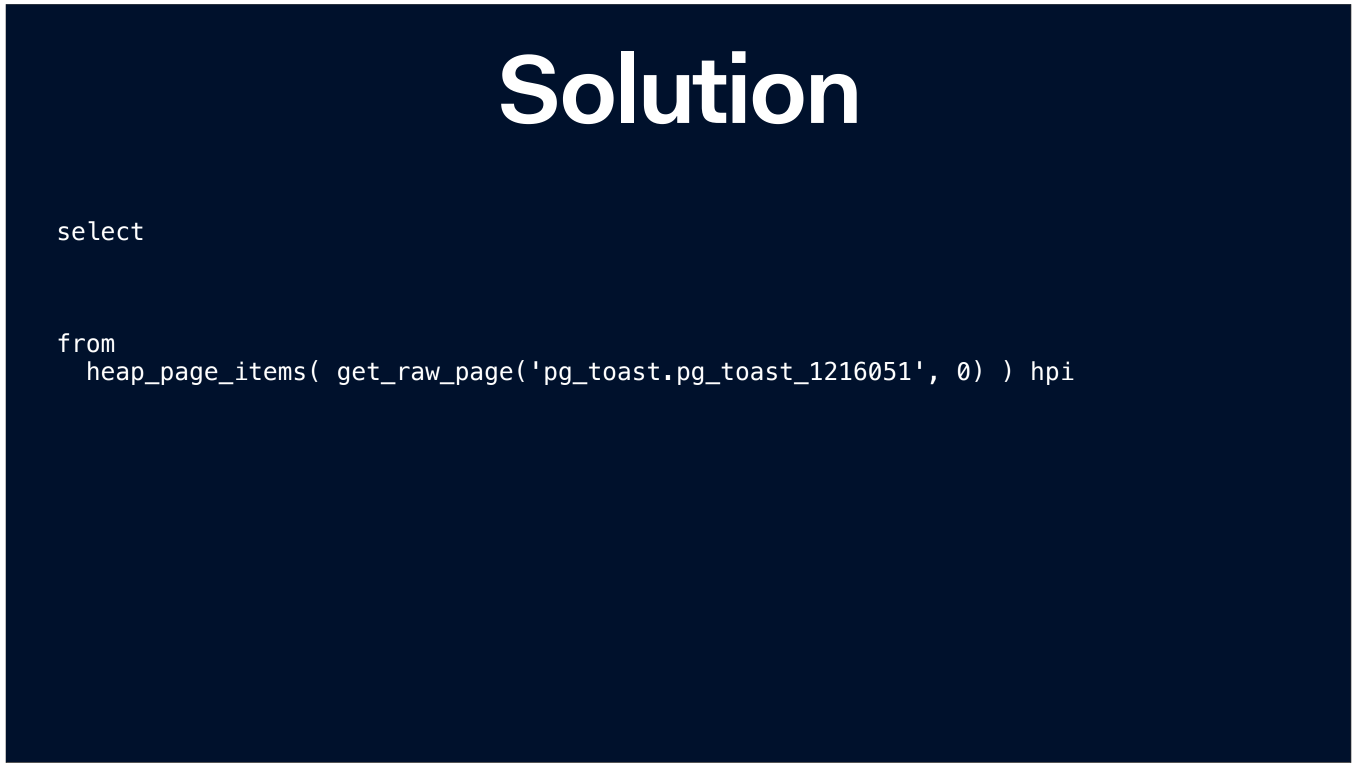
Solution

Toast data exists

Transaction status is unclear

Solution

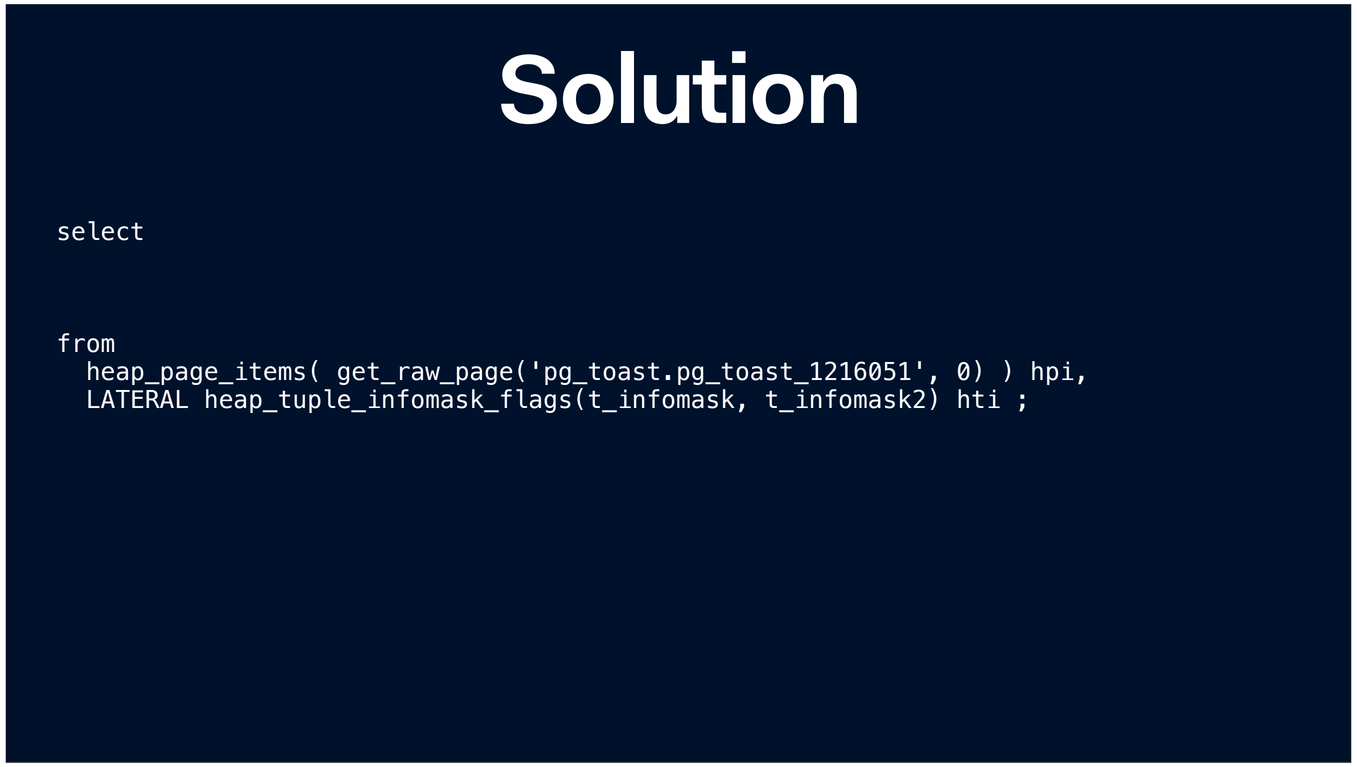
Transaction status



Solution

```
select



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

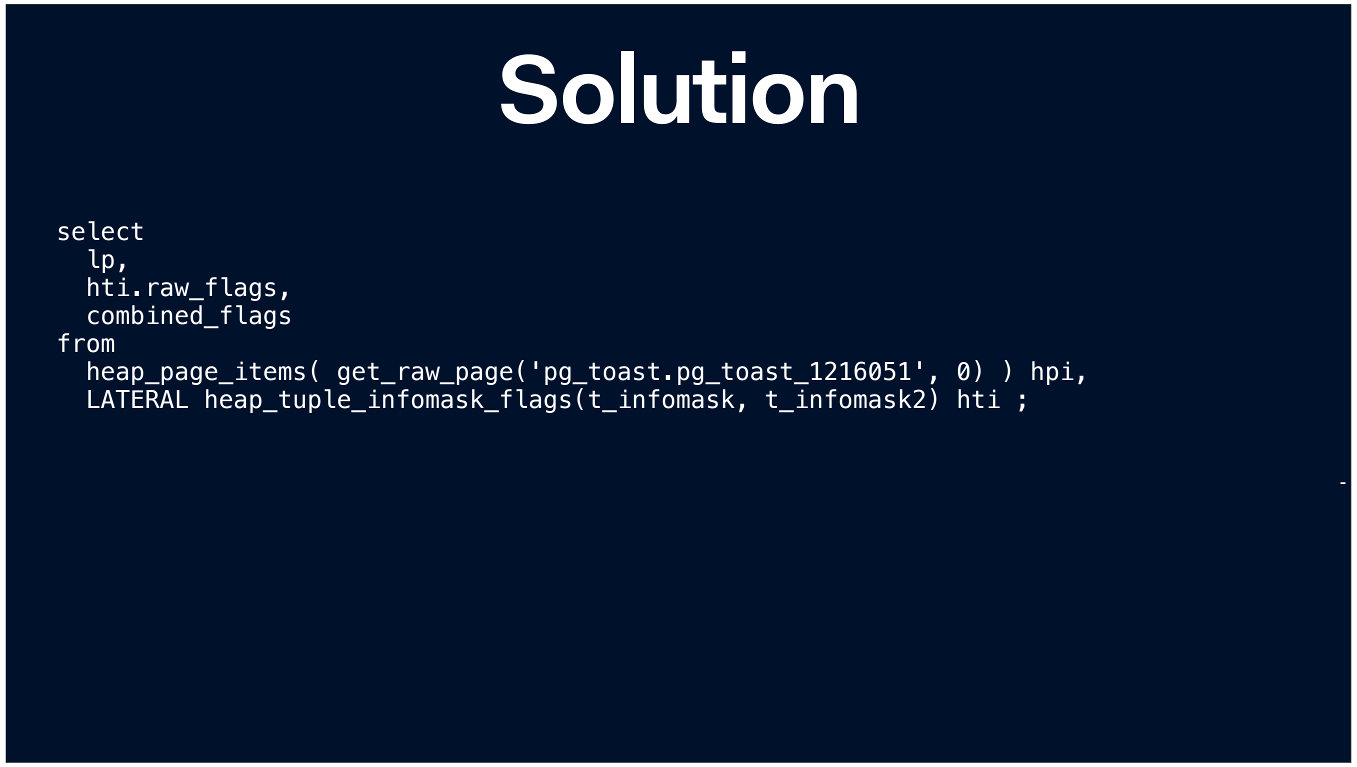
In order to fix the data, we need to understand the transaction status, which is written in t_infomask and can be queried using page_inspect



Solution

```
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



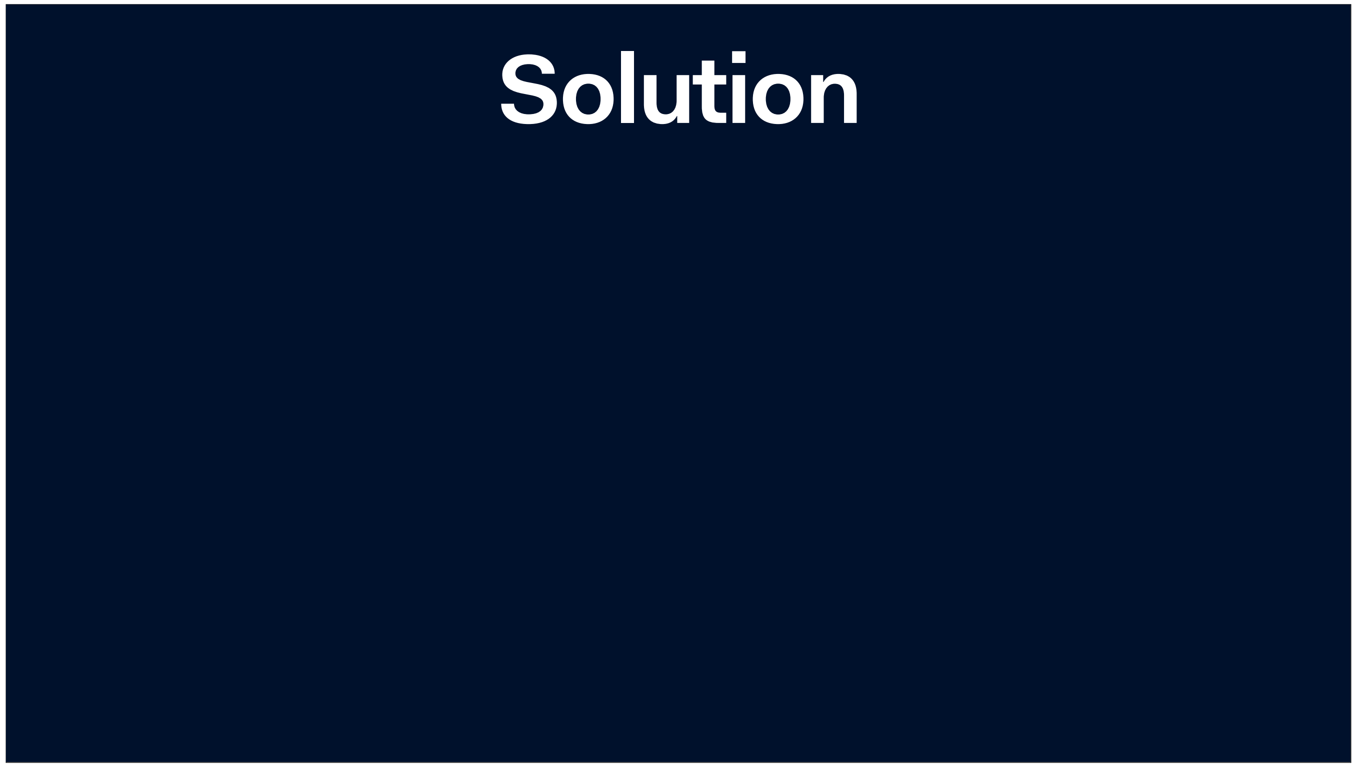
Solution

```
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

Solution

```
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 ;

 lp |                                 raw_flags                                  |   combined_flags
----+----------------------------------------------------------------------------+--------------------
  1 | {HEAP_HASVARWIDTH,HEAP_XMIN_COMMITTED,HEAP_XMAX_INVALID}                   | {}
  2 | {HEAP_HASVARWIDTH,HEAP_XMAX_INVALID}                                       | {}
  3 | {HEAP_HASVARWIDTH,HEAP_XMAX_INVALID}                                       | {}
  4 | {HEAP_HASVARWIDTH,HEAP_XMIN_COMMITTED,HEAP_XMIN_INVALID,HEAP_XMAX_INVALID} | {HEAP_XMIN_FROZEN}
```



Solution

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)
```

Solution

HEAP_HASVARWIDTH	Has Variable-width attributes

Solution

HEAP_HASVARWIDTH	Has Variable-width attributes
HEAP_XMIN_COMMITTED	t_xmin committed

Solution

HEAP_HASVARWIDTH	Has Variable-width attributes
HEAP_XMIN_COMMITTED	t_xmin committed
HEAP_XMIN_INVALID	t_xmin invalid / aborted

Solution

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

Solution

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

Solution

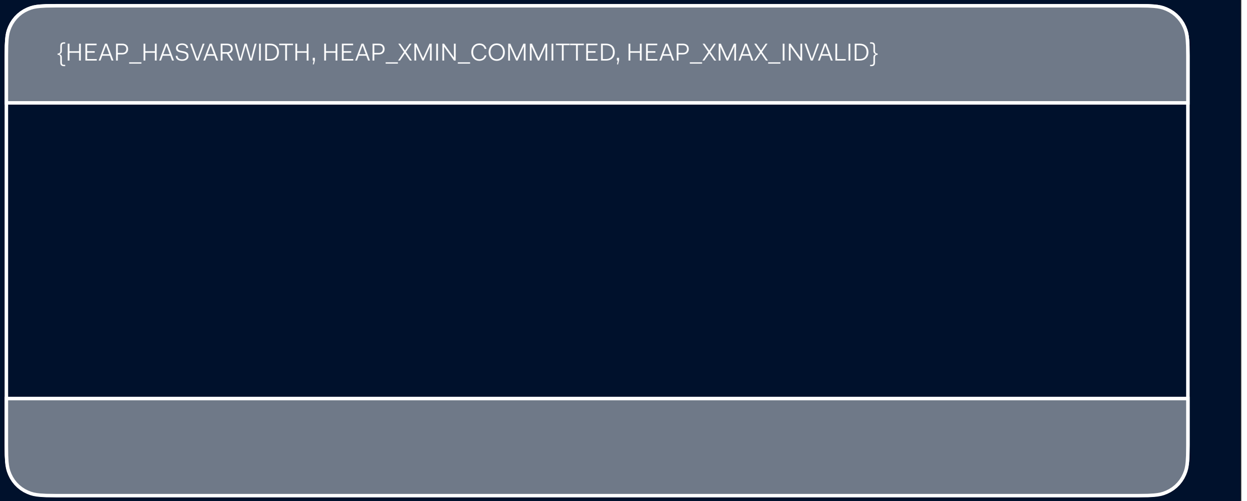
HEAP_HASVARWIDTH	Has Variable-width attributes
HEAP_XMIN_COMMITTED	t_xmin committed
HEAP_XMIN_INVALID	t_xmin invalid / aborted
HEAP_XMAX_COMMITTED	t_xmax committed
HEAP_XMAX_INVALID	t_xmax invalid / aborted
HEAP_KEYS_UPDATED	tuple was updated and key cols modified, or tuple deleted

Solution

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)

Solution

Solution



{HEAP_HASVARWIDTH, HEAP_XMIN_COMMITTED, HEAP_XMAX_INVALID}

Solution

{HEAP_HASVARWIDTH, HEAP_XMIN_COMMITTED, HEAP_XMAX_INVALID}

Committed

Solution

{HEAP_HASVARWIDTH, HEAP_XMIN_COMMITTED, HEAP_XMAX_INVALID}

Committed

Never updated

Solution

{HEAP_HASVARWIDTH, HEAP_XMIN_COMMITTED, HEAP_XMAX_INVALID}

Committed

Never updated

Action: Freeze row

Solution

Solution

{HEAP_HASVARWIDTH, HEAP_XMIN_COMMITTED, HEAP_KEYS_UPDATED}

Might be Committed or aborted

Unclear xmax state

Decide on logic in main table

Solution

{HEAP_HASVARWIDTH, HEAP_XMIN_COMMITTED, HEAP_KEYS_UPDATED}

Might be Committed or aborted

Unclear xmax state

Decide on logic in main table

Action: Logic based on main table transaction status

Solution

Solution

```
{}
```

Empty flags

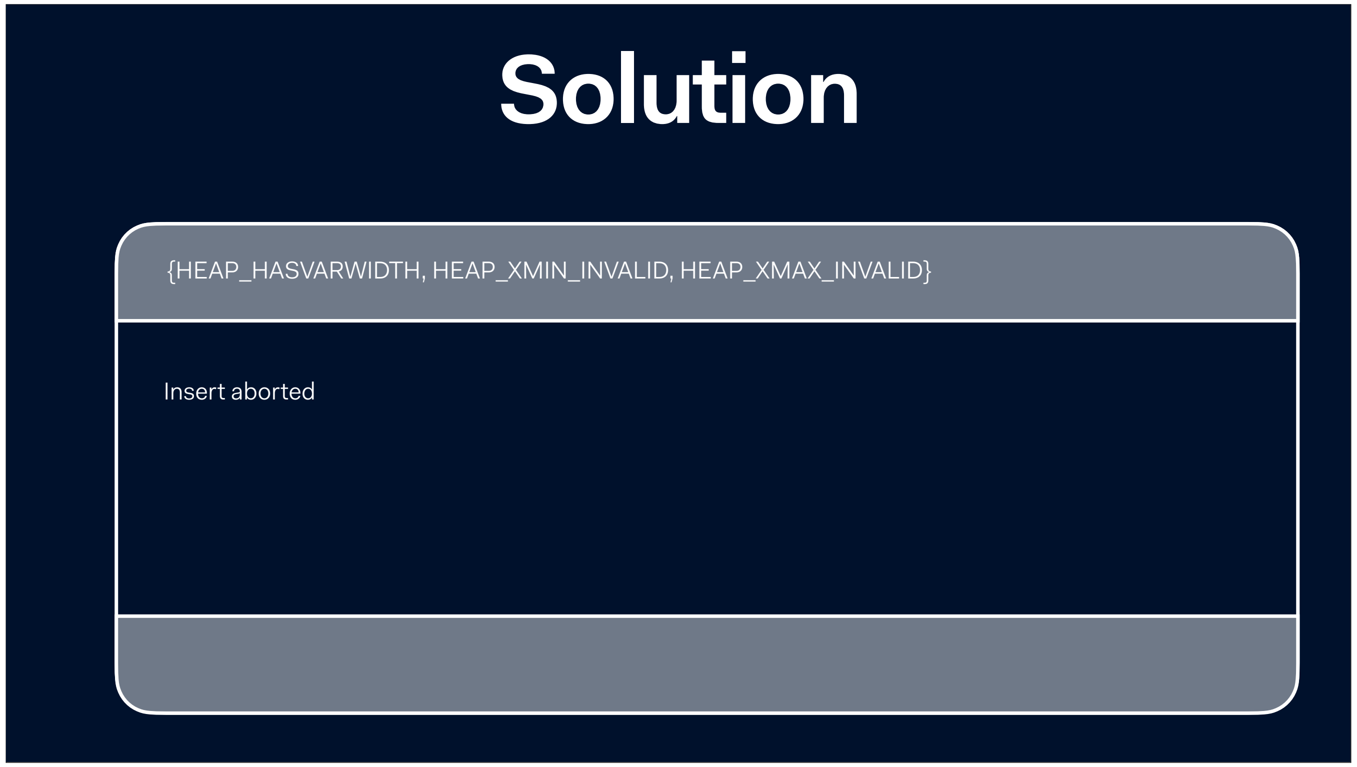
Solution

```
{}
```

Empty flags

Action: Kill row

Solution



Solution

{HEAP_HASVARWIDTH, HEAP_XMIN_INVALID, HEAP_XMAX_INVALID}

Insert aborted

Solution

{HEAP_HASVARWIDTH, HEAP_XMIN_INVALID, HEAP_XMAX_INVALID}

Insert aborted

Action: Kill row

Solution

Solution

{HEAP_HASVARWIDTH, HEAP_XMIN_COMMITTED, HEAP_XMAX_COMMITTED, HEAP_KEYS_UPDATED}

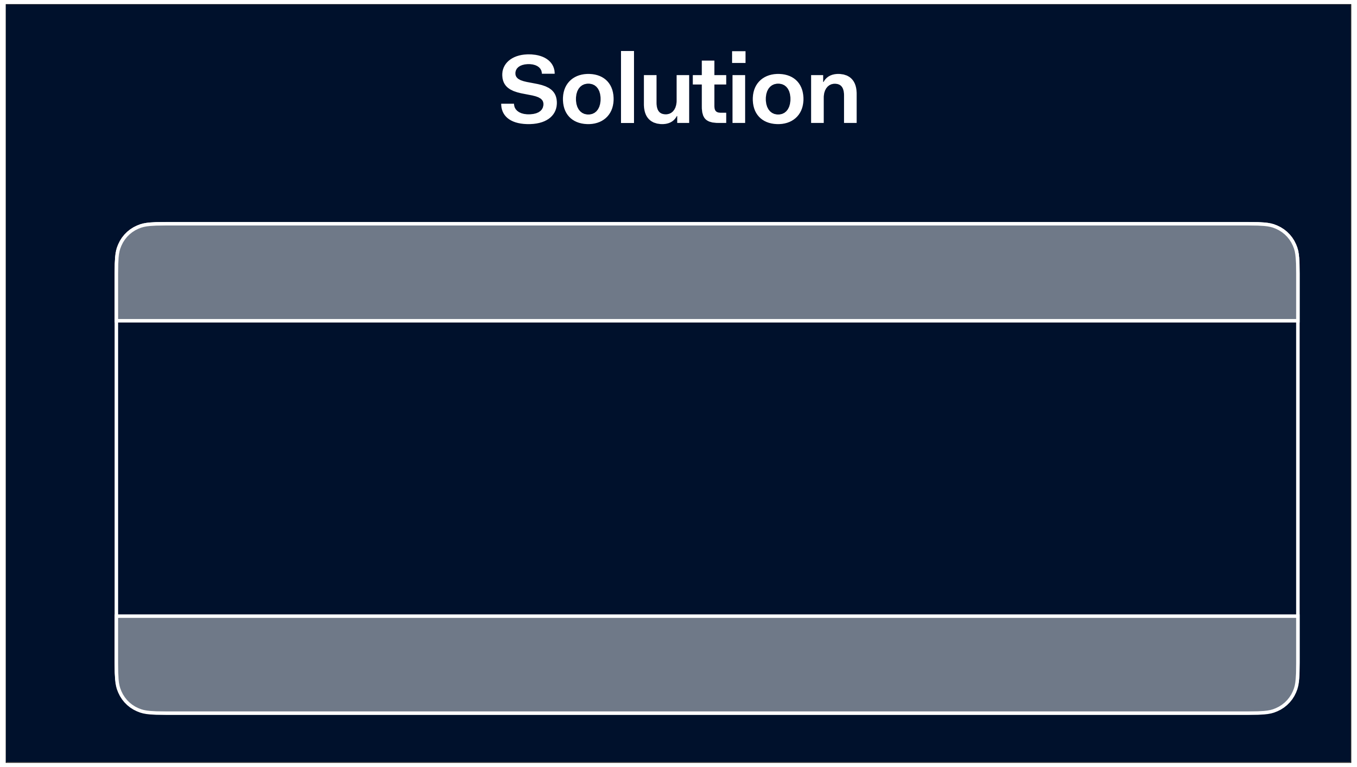
Updated

Solution

{HEAP_HASVARWIDTH, HEAP_XMIN_COMMITTED, HEAP_XMAX_COMMITTED, HEAP_KEYS_UPDATED}

Updated

Action: Kill row



Solution

This tuple doesn't have HEAP_XMIN_COMMITTED, which means we don't know if the transaction committed or not. 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.

Solution

{HEAP_HASVARWIDTH, HEAP_XMAX_INVALID}

heap_xmin_committed not available

Insert has probably aborted

If we are wrong, we delete active data

Solution

{HEAP_HASVARWIDTH, HEAP_XMAX_INVALID}

heap_xmin_committed not available

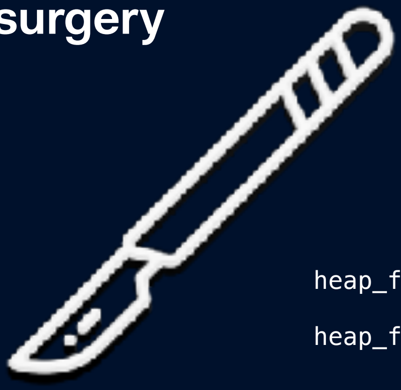
Insert has probably aborted

If we are wrong, we delete active data

Action: Freeze row

Solution

pg_surgery



`heap_force_kill`

`heap_force_freeze`

Summary

Summary

Toast is not trivial to understand

Summary

Toast is not trivial to understand

Extensions

Summary

Toast is not trivial to understand

Extensions

- pageinspect

Summary

Toast is not trivial to understand

Extensions

- pageinspect
- pg_visibility

Summary

Toast is not trivial to understand

Extensions

- pageinspect
- pg_visibility
- pg_surgery

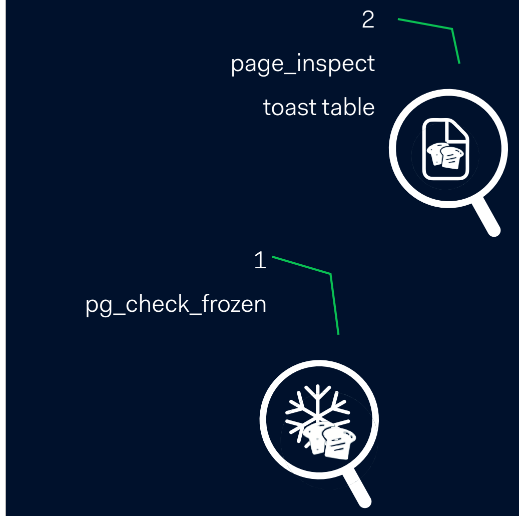
Summary

Summary

1

pg_check_frozen

Summary



2

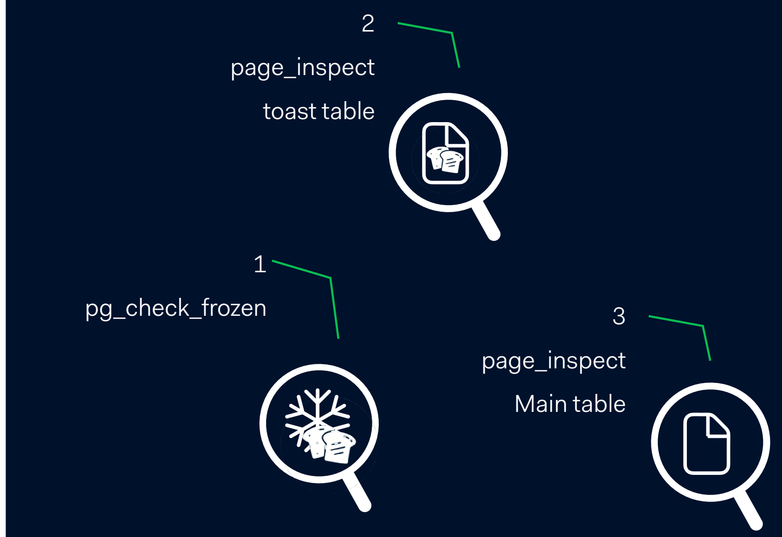
page_inspect

toast table

1

pg_check_frozen

Summary



2

page_inspect

toast table

1

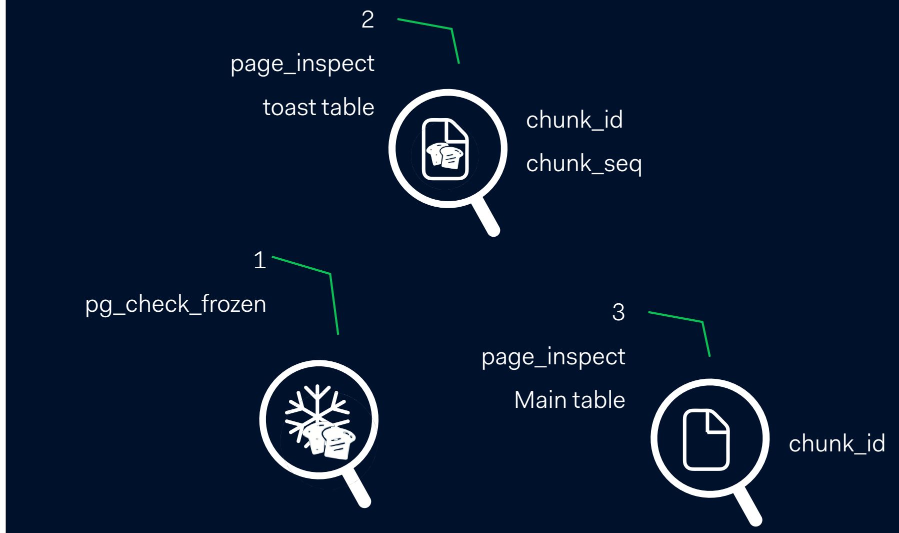
pg_check_frozen

3

page_inspect

Main table

Summary



1

pg_check_frozen

2

page_inspect

toast table

chunk_id

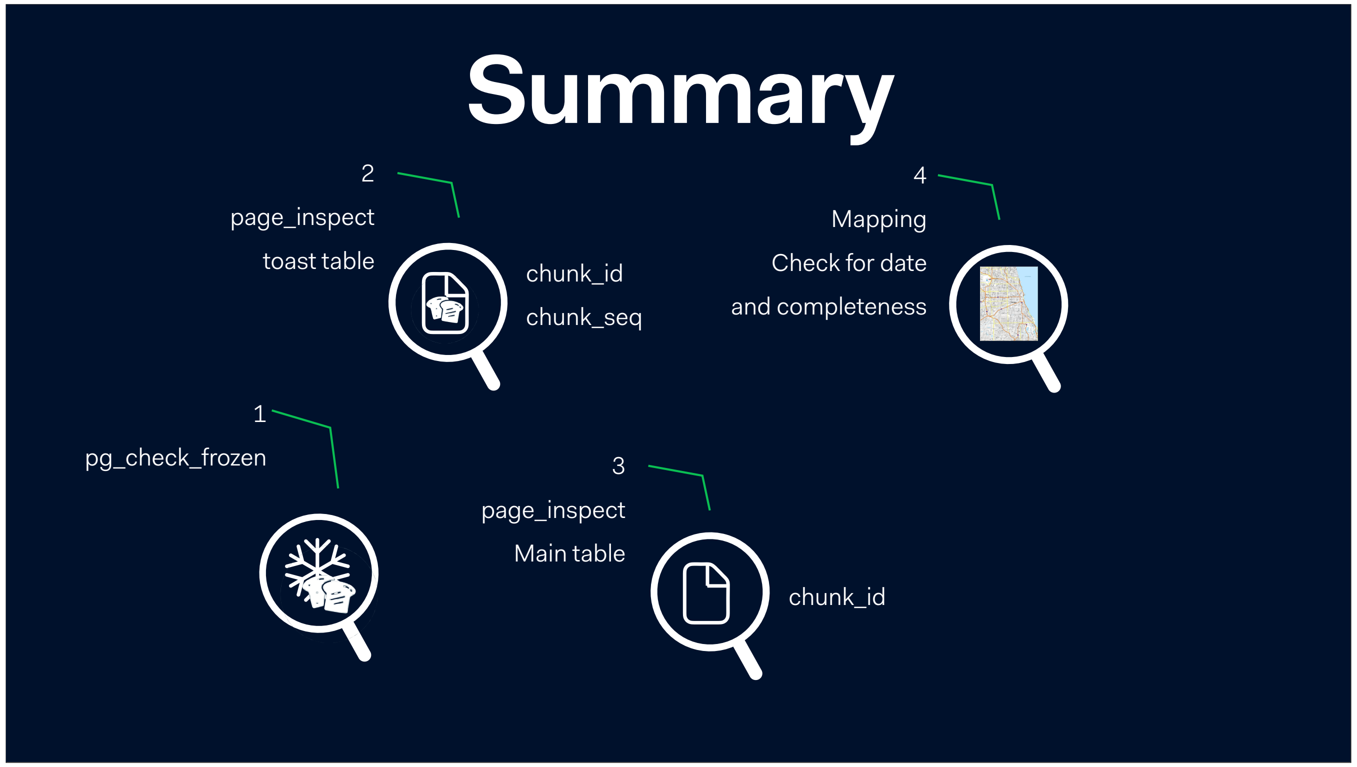
chunk_seq

3

page_inspect

Main table

chunk_id



Summary

1
pg_check_frozen

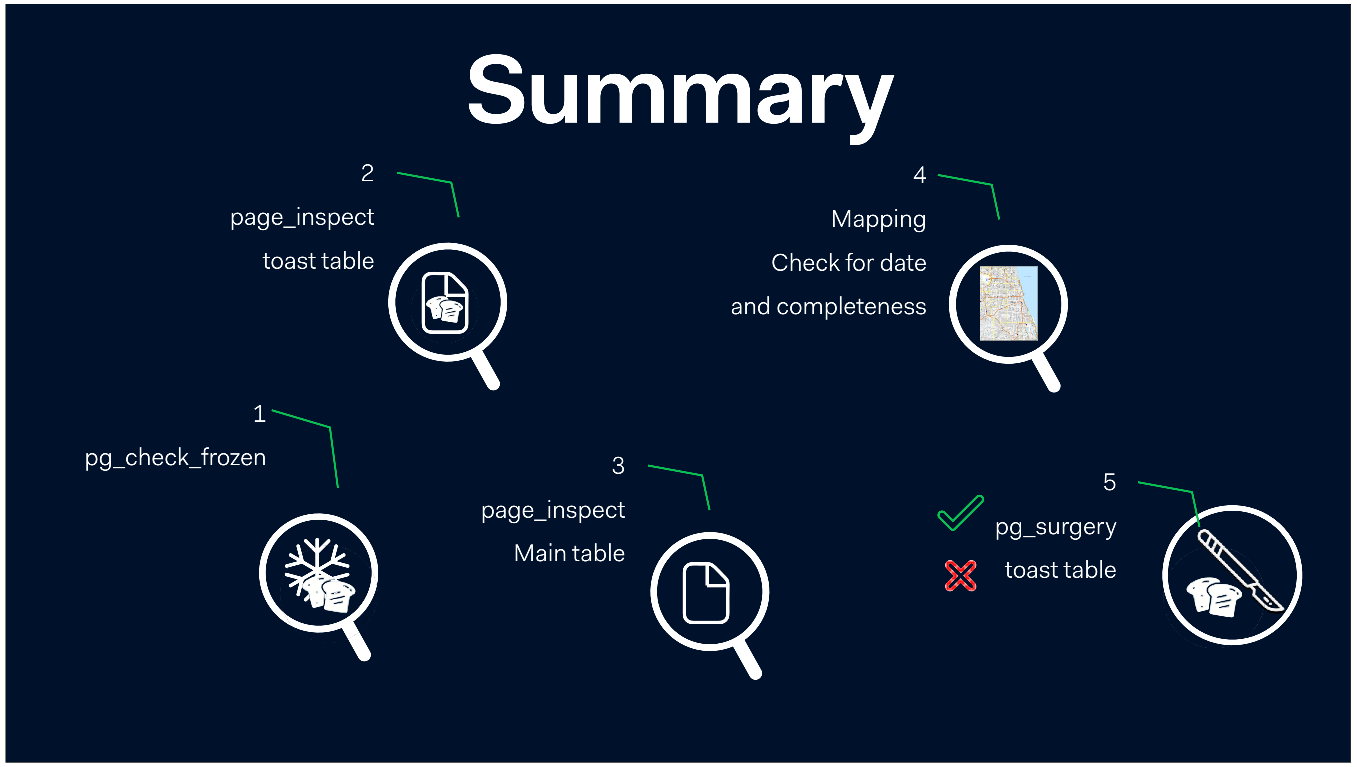
2
page_inspect
toast table

chunk_id
chunk_seq

3
page_inspect
Main table

chunk_id

4
Mapping
Check for date
and completeness



Summary

1

pg_check_frozen

2

page_inspect

toast table

3

page_inspect

Main table

4

Mapping

Check for date

and completeness

5

✓ pg_surgery

✗ toast table

Lessons learned

Detecting

- pg_check_frozen
- amrepair (indexes)

Lessons learned

Prevention

https://www.postgresql.org/message-id/flat/CAM527d_-YmKqFfJmnPsVSeqKC_WHoJVQwxtNvUiV4ju%2B9stDpA%40mail.gmail.com#2b1c4f7c41769f5efbe98d76eb2f3098

Lessons learned

Prevention

- Run rsync without `--size-only` for vm and fsm

Lessons learned

Prevention

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

Lessons learned

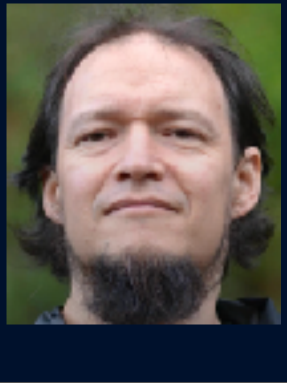
Prevention

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

Special thanks

Cagri Biroglu

Álvaro Herrera
Muñoz



Special thanks

Mark Dilger

Arthur Nascimento

Q & A