credativ

Database in Distress

Testing and Repairing Different Types of Database Corruption

Josef Machytka <josef.machytka@credativ.de>

2025-06-19 - PostgreSQL MeetUp Berlin

credativ GmbH

- Founded 1999 in Jülich, Germany
- Close ties to Open-Source Community
- More than 40 Open-Source experts
- Consulting, development, training, support (3rd-level / 24x7)
- Open-Source infrastructure with Linux, Kubernetes and Proxmox
- Open-Source databases with PostgreSQL
- DevSecOps with Ansible, Puppet, Terraform and others
- Since 2025 independent owner-managed company again

Josef Machytka

credativ

- Professional Service Consultant - PostgreSQL specialist at credativ GmbH
- 30+ years of experience with different databases
- PostgreSQL (13y), BigQuery (7y), Oracle (15y), MySQL (12y), Elasticsearch (5y), MS SQL (5y)
- 10+ years of experience with Data Ingestion pipelines, Data Analysis, Data Lake and Data Warehouse
- 2+ years of practical experience with different LLMs / AI including their architecture and principles
- From Czechia, living now 11 years in Berlin

  - LinkedIn: linkedin.com/in/josef-machytka
  - ResearchGate: researchgate.net/profile/Josef-Machytka
  - Academia.edu: academia.edu/JosefMachytka
  - Medium: medium.com/@josef.machytka
  - sessionize: sessionize.com/josefmachytka

Table of contents

AI images created by the author
using ChatGPT DALL-E

credativ

Corruption is often hidden

How to Recognize Data Corruption

- PostgreSQL can use data block checksums to detect corruption
- But in PostgreSQL 17 or older checksums are not enabled due to performance impact
- Corruption is therefore usually shown as errors during pg_dump backup

```
pg_dump: error: query to get data of sequence "test_table_bytea_id2_seq" returned 0 rows (expected 1)
```

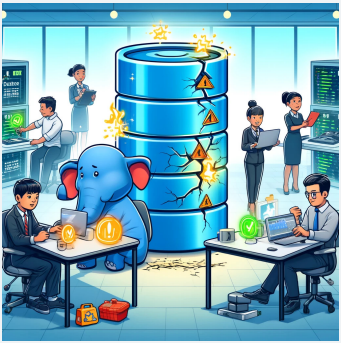
```
pg_dump: error: query failed: ERROR:  invalid page in block 0 of relation base/16384/66427
pg_dump: error: query was: SELECT last_value, is_called FROM public.test_table_bytea_id_seq
```

```
pg_dump: error: Dumping the contents of table "test_table_bytea" failed: PQgetResult() failed.
pg_dump: error: Error message from server: ERROR:  could not access status of transaction 3353862211
DETAIL:  Could not open file "pg_xact/0C7E": No such file or directory.
pg_dump: error: The command was: COPY public.test_table_bytea (id, id2, id3, description, data) TO stdout;
```

```
pg_dump: error: Dumping the contents of table "test_table_bytea" failed: PQgetCopyData() failed.
pg_dump: error: Error message from server: server closed the connection unexpectedly
        This probably means the server terminated abnormally
        before or while processing the request.
pg_dump: error: The command was: COPY public.test_table_bytea (id, id2, id3, description, data) TO stdout;
```

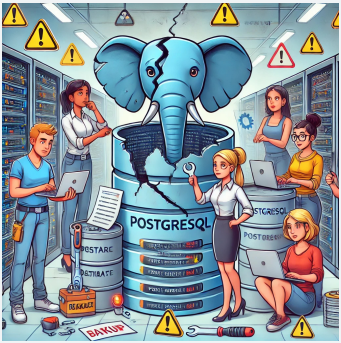
Corruption is often hidden

- Without checksums, some types of corruption may not be detected
- Backup tools which do not directly select records hide corruption
- pg_basebackup copies data files as they are, without checking
- To reveal corruption we need to select data from relations
- Problems are usually shown as errors during pg_dump backup
- Because pg_dump extracts content of tables and sequences



Corruption is often hidden

- Without checksums we can have corrupted data without noticing
- Damage in the old records, which are not selected any more
- Damage in the auditing or history tables
- Damage in the indexes which are not used
- Autovacuum can fail on some types of corruption, but we might not notice
- Damage can be in already frozen tuples, not vacuumed any more



What about Checksums?

- PostgreSQL can use data block checksums to detect corruption
- But in PostgreSQL 17 or older checksums are not enabled due to performance impact
- Can be activated later with server application pg_checksums
- But it requires downtime and we must be sure data are not corrupted already
- We also have parameter ”ignore_checksum_failures”
- PostgreSQL 18 sets checksums ON by default in initdb

```
initdb: Change default to using data checksums.
author  Peter Eisentraut <peter@eisentraut.org>
        Wed, 16 Oct 2024 06:45:09 +0000 (08:45 +0200)
committer   Peter Eisentraut <peter@eisentraut.org>
                Wed, 16 Oct 2024 06:48:10 +0000 (08:48 +0200)

Checksums are now on by default.  They can be disabled by the previously added option --no-data-checksums.

Author: Greg Sabino Mullane <greg@turnstep.com>
Reviewed-by: Nathan Bossart <nathandbossart@gmail.com>
Reviewed-by: Peter Eisentraut <peter@eisentraut.org>
Reviewed-by: Daniel Gustafsson <daniel@yesql.se>
Discussion: https://www.postgresql.org/message-id/flat/CAKAnmmKwiMHik5AHmBEdf5vqzbOBbcwEPHo4-PioWeAbzwcTOQ@mail.gmail.com
```

How detection of corruption with checksums look like?

- Checksums are calculated for each data block, stored in the header
- When data block is read, checksum is calculated again and compared with stored value
- If values do not match, PostgreSQL raises error
- Damaged blocks can be skipped and ”zeroed out” by setting ”zero_damaged_pages=ON”

```
-- zero_damaged_pages=OFF --> Query stops on the first checksum error
test=# select * from test_table_bytea;
WARNING: page verification failed, calculated checksum 19601 but expected 152
ERROR: invalid page in block 0 of relation base/16384/16402

-- zero_damaged_pages=ON --> Query skips damaged blocks and continues
test=# select * from pg_toast.pg_toast_17453;
WARNING: page verification failed, calculated checksum 29668 but expected 57724
WARNING: invalid page in block 204 of relation base/16384/17464; zeroing out page
WARNING: page verification failed, calculated checksum 63113 but expected 3172
WARNING: invalid page in block 222 of relation base/16384/17464; zeroing out page
WARNING: page verification failed, calculated checksum 59128 but expected 3155
WARNING: invalid page in block 260 of relation base/16384/17464; zeroing out page
WARNING: page verification failed, calculated checksum 14957 but expected 16239
....
```

How detection of corruption with checksums look like?

- Or errors can be ignored setting ”ignore_checksum_failures=ON”
- This setting just skips errors, corrupted pages are not zeroed out

```
-- ignore_checksum_failure="off";  --- Query stops on checksum error
test=# select * from pg_toast.pg_toast_17453;
WARNING: page verification failed, calculated checksum 19601 but expected 152
ERROR: invalid page in block 0 of relation base/16384/16402

-- ignore_checksum_failure="on";  --- Query skips checksum errors
test=# select * from pg_toast.pg_toast_17453;
WARNING:  page verification failed, calculated checksum 29668 but expected 57724
WARNING:  page verification failed, calculated checksum 63113 but expected 3172
WARNING:  page verification failed, calculated checksum 59128 but expected 3155
WARNING:  page verification failed, calculated checksum 14957 but expected 16239
ERROR:  could not access status of transaction 3088756928
DETAIL:  Could not open file "pg_xact/0B81": No such file or directory.
```

Any PostgreSQL Object Can Be Corrupted

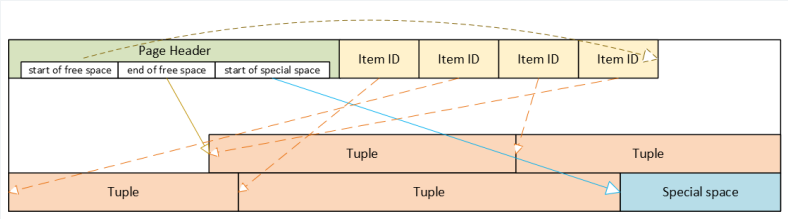
- Main tables, TOAST tables
- Sequences
- Indexes
- Free space maps
- Visibility maps
- In this talk I focus on tables, and sequences
- I created ”Corruption Simulator” in python
- Code ”surgically” damages selected parts of data block

credativ

Anatomy of Data Corruption

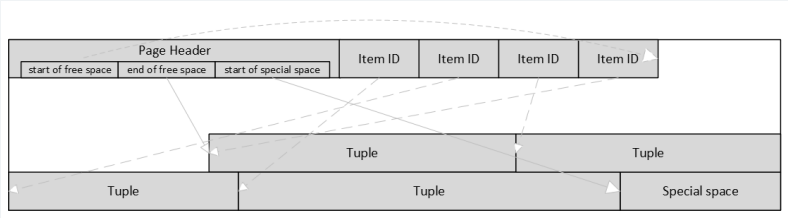
Anatomy of Data Block

- In core PostgreSQL everything is stored as a heap table
- Within 8 KB data blocks of the same topology
- Data block has 5 parts: Header, ItemIDs array, Free space, Tuples, Special space
- Objects can have different content in the ”special space” at the end of the block



Different Types of Data Corruption

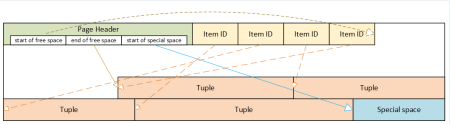
- Depending on damaged part, different errors will be reported
- Corrupted header means whole block is lost
- Corrupted ItemIDs array means no access to tuples
- Corrupted tuples mean random values in the tuples and their system columns
- Corrupted special space breaks some other relations



Corrupted Header

- Header occupies first 24 bytes of data block
- Corrupted header means the whole content of block is lost
- ERROR: invalid page in block 285 of relation base/16384/29724
- Only this error can be ”cured” by setting ”zero_damaged_pages = on”
- Blocks with corrupted header are ”zeroed” in memory and skipped
- VACUUM can remove these blocks
- Nothing to diagnose here, all selects fail

Field	Type	Length	Description
pd_lsn	PageXLogRecPtr	8 bytes	LSN: next byte after last byte of WAL record for last change to this page
pd_checksum	uint16	2 bytes	Page checksum
pd_flags	uint16	2 bytes	Flag bits
pd_lower	LocationIndex	2 bytes	Offset to start of free space
pd_upper	LocationIndex	2 bytes	Offset to end of free space
pd_special	LocationIndex	2 bytes	Offset to start of special space
pd_pagesize_version	uint16	2 bytes	Page size and layout version number information
pd_prune_xid	TransactionId	4 bytes	Oldest unpruned XMAX on page, or zero if none



Corrupted Header - look into code

- Looking at the branch "REL_17_STABLE" in the source code
- Error message "invalid page in block xx of relation ..."
- From src/backend/catalog/storage.c - "RelationCopyStorage"
- If "PageIsVerifiedExtended" (src/backend/storage/page/bufpage.c) returns "false"

Field	Type	Length	Description
pd_lsn	PageXLogRecPtr	8 bytes	LSN: next byte after last byte of WAL record for last change to this page
pd_checksum	uint16	2 bytes	Page checksum
pd_flags	uint16	2 bytes	Flag bits
pd_lower	LocationIndex	2 bytes	Offset to start of free space
pd_upper	LocationIndex	2 bytes	Offset to end of free space
pd_special	LocationIndex	2 bytes	Offset to start of special space
pd_pagesize_version	uint16	2 bytes	Page size and layout version number information
pd_prune_xid	TransactionId	4 bytes	Oldest unpruned XMAX on page, or zero if none

```
        /*
         * The following checks don't prove the header is correct, only that
         * it looks sane enough to allow into the buffer pool. Later usage of
         * the block can still reveal problems, which is why we offer the
         * checksum option.
         */
        if ((p->pd_flags & ~PD_VALID_FLAG_BITS) == 0 &&
            p->pd_lower <= p->pd_upper &&
            p->pd_upper <= p->pd_special &&
            p->pd_special <= BLCKSZ &&
            p->pd_special == MAXALIGN(p->pd_special))
            header_sane = true;

        if (header_sane && !checksum_failure)
            return true;
```

Corrupted Header - look into code

- Let's look at PD_VALID_FLAG_BITS
- Defined in src/include/storage/bufpage.h

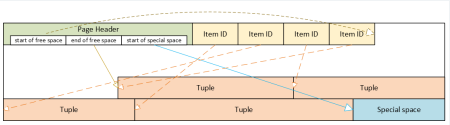
Field	Type	Length	Description
pd_lsn	PageXLogRecPtr	8 bytes	LSN: next byte after last byte of WAL record for last change to this page
pd_checksum	uint16	2 bytes	Page checksum
pd_flags	uint16	2 bytes	Flag bits
pd_lower	LocationIndex	2 bytes	Offset to start of free space
pd_upper	LocationIndex	2 bytes	Offset to end of free space
pd_special	LocationIndex	2 bytes	Offset to start of special space
pd_pagesize_version	uint16	2 bytes	Page size and layout version number information
pd_prune_xid	TransactionId	4 bytes	Oldest unpruned XMAX on page, or zero if none

```
/*
 * pd_flags contains the following flag bits.  Undefined bits are initialized
 * to zero and may be used in the future.
 *
 * PD_HAS_FREE_LINES is set if there are any LP_UNUSED line pointers before
 * pd_lower.  This should be considered a hint rather than the truth, since
 * changes to it are not WAL-logged.
 *
 * PD_PAGE_FULL is set if an UPDATE doesn't find enough free space in the
 * page for its new tuple version; this suggests that a prune is needed.
 * Again, this is just a hint.
 */
#define PD_HAS_FREE_LINES   0x0001  /* are there any unused line pointers? */
#define PD_PAGE_FULL        0x0002  /* not enough free space for new tuple? */
#define PD_ALL_VISIBLE      0x0004  /* all tuples on page are visible to
                                     * everyone */

#define PD_VALID_FLAG_BITS  0x0007  /* OR of all valid pd_flags bits */
```

Corrupted Item IDs Array

- Contains 4 bytes pointers to the tuples (offset and length)
- Corrupted item ids array means no access to tuples
- Offset and Length contain random, often bigger than 8192



- ERROR: invalid memory alloc request size 18446744073709551594
- DEBUG: server process (PID 76) was terminated by signal 11: Segmentation fault

```
SELECT lp, lp_off, lp_flags, lp_len, t_xmin, t_xmax, t_field3, t_ctid, t_infomask2, t_infomask, t_hoff, t_bits, t_oid, substr(t_data::text,1,50) as t_data
FROM heap_page_items(get_raw_page('public.test_table_bytea', 7));
```

lp	lp_off	lp_flags	lp_len	t_xmin	t_xmax	t_field3	t_ctid	t_infomask2	t_infomask	t_hoff	t_bits	t_oid	t_data
1	7936	1	252	29475	0	0	(7,1)	5	2310	24			\x01010000010100000101000018030000486f742073656520
2	7696	1	236	29476	0	0	(7,2)	5	2310	24			\x02010000020100000201000d802000043756c747572616c
3	7504	1	189	29477	0	0	(7,3)	5	2310	24			\x03010000030100000301000001c020000446f6f7220726563
4	7368	1	132	29478	0	0	(7,4)	5	2310	24			\x04010000040100000401000009d4d6f76656d656e74207374
5	7128	1	238	29479	0	0	(7,5)	5	2310	24			\x05010000050100000501000e0020000426f617264207065
6	6872	1	249	29480	0	0	(7,6)	5	2310	24			\x06010000060100000601000000c030000576866f6c6520616c
7	6648	1	219	29481	0	0	(7,7)	5	2310	24			\x07010000070100000701000094020000416756e6379206d

lp	lp_off	lp_flags	lp_len	t_xmin	t_xmax	t_field3	t_ctid	t_infomask2	t_infomask	t_hoff	t_bits	t_oid	t_data
1	6577	1	28310										
2	20113	3	13097										
3	1273	1	28308										
4	17972	0	15077										
5	11161	2	28274										

Corrupted Item IDs Array - Segmentation Fault

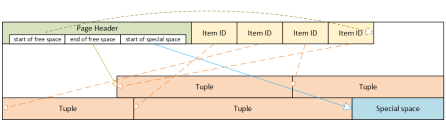
- Corrupted Item IDs Array causes memory allocation errors
- Worst case - session crashes with Segmentation fault
- Crash is caught by src/backend/postmaster/postmaster.c
- ”HandleChildCrash” -> ”LogChildExit”

```
if (!EXIT_STATUS_0(exitstatus))
    activity = pgstat_get_crashed_backend_activity(pid,
                                                   activity_buffer,
                                                   sizeof(activity_buffer));
...

ereport(lev,
/*------
    translator: %s is a noun phrase describing a child process, such as
    "server process" */
        (errmsg("%s (PID %d) was terminated by signal %d: %s",
                procname, pid, WTERMSIG(exitstatus),
                pg_strsignal(WTERMSIG(exitstatus))),
            activity ? errdetail("Failed process was running: %s", activity) : 0));
```

Corrupted Tuples

- Corrupted tuples mean random values in the columns
- Each tuple has header with system columns - 27 bytes
- Especially corrupted xmin, xmax and ctid are critical



Field	Type	Length	Description
t_xmin	TransactionId	4 bytes	insert XID stamp
t_xmax	TransactionId	4 bytes	delete XID stamp
t_cid	CommandId	4 bytes	insert and/or delete CID stamp (overlays with t_xvac)
t_xvac	TransactionId	4 bytes	XID for VACUUM operation moving a row version
t_ctid	ItemPointerData	6 bytes	current TID of this or newer row version
t_infomask2	uint16	2 bytes	number of attributes, plus various flag bits
t_infomask	uint16	2 bytes	various flag bits
t_hoff	uint8	1 byte	offset to user data

Corrupted Tuples

- 58P01 - could not access status of transaction 3047172894
- XX000 - MultiXactId 1074710815 has not been created yet – apparent wraparound
- ERROR: invalid memory alloc request size 18446744073709551594
- WARNING: Concurrent insert in progress within table ”test_table_bytea”

```
SELECT lp, lp_off, lp_flags, lp_len, t_xmin, t_xmax, t_field3, t_ctid, t_infomask2, t_infomask, t_hoff, t_bits, t_oid, substr(t_data::text,1,50) as t_data
FROM heap_page_items(get_raw_page('pg_toast.pg_toast_64234', 1655));

 lp | lp_off | lp_flags | lp_len | t_xmin | t_xmax | t_field3 |  t_ctid   | t_infomask2 | t_infomask | t_hoff | t_bits | t_oid |                    t_data
----+--------+----------+--------+--------+--------+----------+-----------+-------------+------------+--------+--------+-------+-----------------------------------------------
  1 |   6160 |        1 |   2032 |  29377 |      0 |        0 | (1655,1)  |           3 |       2050 |     24 |        |       | \x9cfb00000c000000401f0000bf5bcb49edc7eab0ecaebc11
  2 |   4128 |        1 |   2032 |  29377 |      0 |        0 | (1655,2)  |           3 |       2050 |     24 |        |       | \x9cfb00000d000000401f0000bd871bee922f63de0a19e7f1
  3 |   2096 |        1 |   2032 |  29377 |      0 |        0 | (1655,3)  |           3 |       2050 |     24 |        |       | \x9cfb00000e000000401f000074d1fb7bde41c51d18322d85
  4 |     64 |        1 |   2032 |  29377 |      0 |        0 | (1655,4)  |           3 |       2050 |     24 |        |       | \x9cfb00000f000000401f0000f913a066a35506e5c5293fee
(4 rows)

SELECT lp, lp_off, lp_flags, lp_len, t_xmin, t_xmax, t_field3, t_ctid, t_infomask2, t_infomask, t_hoff, t_bits, t_oid, substr(t_data::text,1,50) as t_data
FROM heap_page_items(get_raw_page('pg_toast.pg_toast_64234', 1654));

 lp | lp_off | lp_flags | lp_len |   t_xmin   |   t_xmax   |  t_field3   |       t_ctid       | t_infomask2 | t_infomask | t_hoff | t_bits | t_oid |                    t_data
----+--------+----------+--------+------------+------------+-------------+--------------------+-------------+------------+--------+--------+-------+-----------------------------------------------
  1 |   6160 |        1 |   2032 |  534805867 |  873710472 | -1291348379 | (2395715003,28289) |       19277 |      32958 |     53 |        |       | \xfa80109a61a85e45de4ab580a26
  2 |   4128 |        1 |   2032 | 1026820500 | 1776295209 |  2142400425 | (1412156050,62855) |       11710 |      65140 |    253 |        |       | \xb9ebb3e852ca77a5195bc768469
  3 |   2096 |        1 |   2032 | 1509174647 | 1612155862 |   -79064812 | (3682366356,39443) |        2403 |      63137 |     97 |        |       | \x7e18ac3b8504440b7938764232
  4 |     64 |        1 |   2032 |      29377 |          0 |           0 | (1654,4)           |           3 |       2050 |     24 |        |       | \x9cfb00000b000000401f000085
(4 rows)
```

Corrupted Tuples of Toast Table

- Damaged toast table causes additional errors in the main table

- XX000 - unexpected chunk number 19 (expected 16) for toast value 29685 in pg_toast_29580
- XX000 - unexpected chunk number 1873032786 (expected 4) for toast value 29595 in pg_toast_29580
- XX000 - unexpected chunk number -556107646 (expected 20) for toast value 29611 in pg_toast_29580
- XX000 - found toasted toast chunk for toast value 29707 in pg_toast_29580

Corrupted Sequence

- Sequence is non-transactional table with 1 data block and 1 row
- Row has hidden system columns and 3 normal columns, one of them ”last_value”

- SELECT nextval('<sequence>');
- Corrupted Header -> ERROR: invalid page in block 0 of relation base/16384/64228
- Corrupted Item IDs Array / Tuples:
  - ERROR: bad magic number in sequence ”<sequence>”: 00000017
  - ERROR: invalid memory alloc request size 18446744073709551477

- SELECT * FROM <sequence>;
- Corrupted Header -> ERROR: invalid page in block 0 of relation base/16384/64228
- Corrupted Item IDs Array / Tuples -> returns 0 rows

Corrupted Primary Key Btree Index

- SELECT * FROM public.test_table_bytea WHERE id = xx;
- ERROR: index ”test_table_bytea_pkey” is not a btree

- Corrupted Page Header:
  - XX001-invalid page in block 2 of relation base/16384/66449
- Corrupted Item IDs Array:
  - XX000-invalid memory alloc request size 18446744073709551593
  - After several tests connection crashed with Segmentation fault
- Corrupted Tuples:
  - XX000-could not open file ”base/16384/66449.1” (target block 4092726194): previous segment is only 9 blocks

```
 blkno | type | live_items | dead_items | avg_item_size | page_size | free_size | btpo_prev | btpo_next | btpo | btpo_flags
-------+------+------------+------------+---------------+-----------+-----------+-----------+-----------+------+------------
     1 | l    |        367 |          0 |            16 |      8192 |       808 |         0 |         2 |    0 |          1
(1 row)

 blkno | type | live_items | dead_items | avg_item_size | page_size | free_size | btpo_prev  | btpo_next  |    btpo    | btpo_flags
-------+------+------------+------------+---------------+-----------+-----------+------------+------------+------------+------------
     1 | r    |        273 |         94 |          1407 |      8192 |       808 | -487937076 | -202241102 | -939632903 |      38890
(1 row)
```

Output from pageinspect extension - bt_page_stats

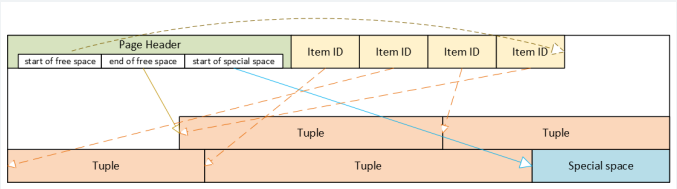
credativ

Can we cure corrupted data?

Can we cure corrupted data?

- Corrupted data block contains random values
- Corrupted header means no access to the block at all
- Corrupted ItemIDs Array means no access to the tuples
- Corrupted tuples mean random values in the columns

- Can we cure corrupted data?
- NO -> if we do not have relevant backups
- We can only remove corrupted data / salvage good data



How to remove corrupted data

- In PostgreSQL 17 or older, without checksums:
- We can easily remove only data pages with corrupted headers
- Parameter ”zero_damaged_pages = ON”
- Blocks with corrupted header are ”zeroed” in memory and skipped
- VACUUM can remove these blocks
- Other types of corruption cannot be so easily removed
- We can only salvage good data or try to use pg_surgery functions

How to salvage good data

- Without checksums, we must salvage good data from corrupted tables
- Script below is only an example, see more details in the talk:
- Laurenz Albe: How to corrupt your database (and how to deal with data corruption) (PGConf.EU 2023)

```
do $$
declare
    i record;
    d record;
begin
for i in (select id from public.test_table_bytea) loop
    begin
        select * into d from public.test_table_bytea where id=i.id;
        insert into public.test_table_bytea_new values (...) values (d....);
        commit;
    exception when  others then
        raise notice 'id=%', i.id
        insert into public.test_table_bytea_broken_ids (id) values (i.id);
    end;
end loop;
end;
```

Removing bad data with pg_surgery

- Extension pg_surgery is ”unsafe by design”
- Allows low-level operations on data blocks
- Function ”heap_force_kill”
- Idea is to forcibly mark damaged tuples as dead
- So they can be skipped by PostgreSQL without errors
- This operation can be much faster than salvage of good data
- But again, it is ”unsafe by design”

How to repair other corrupted objects

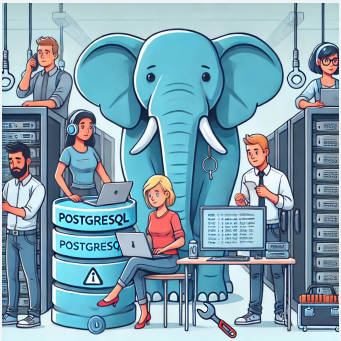
- Damaged indexes can be rebuilt with REINDEX
- Presuming of course data in the table is OK
- Damaged sequence must be replaced with a new one
- And the ”last_value” must be set based on the table
- Damaged system tables might require new installation

credativ

Summary

Key Takeaways

- Data blocks checksums allow immediate detection of corruption
- Corrupted data blocks can be ”zeroed out” and removed
- Without checksums, PostgreSQL can detect only corrupted headers
- PostgreSQL 18 sets checksums ON by default in initdb
- But old installations will most likely stay without checksums
- Adding checksums later is possible but requires downtime
- People can add checksums to already corrupted pages without knowing

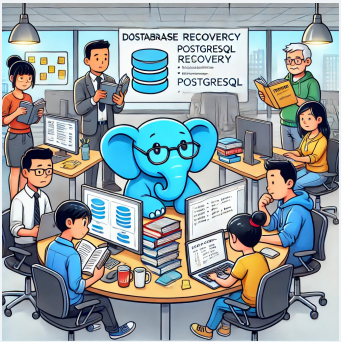


Key Takeaways

- We need additional sanity checks to prevent ”Segmentation Fault” crashes
- In ItemIDs Array Offset and Tuple length cannot exceed page size
- In tuples second part of ctid cannot be larger than few hundred
- Huge xmin/xmax values are suspicious or even impossible
- Setting for ignoring rows with clearly corrupted xmin/xmax fields
- Ridiculous content of system columns mean tuple is lost anyway
- pg_dump should be able to skip corrupted records

Resources

- PostgreSQL talks:
  - PostgreSQL Database Corruption (YouTube Playlist)
  - Data Corruption Bugs: Diagnosis and Lessons (PGConf.dev 2024)
  - Laurenz Albe: How to corrupt your database (and how to deal with data corruption) (PGConf.EU 2023)
- Articles:
  - PostgreSQL Wiki: Corruption
  - Garrett's Blog: Salvaging a Corrupted Table from PostgreSQL
  - StackOverflow: Repair corrupt database PostgreSQL
  - Salvaging a Corrupted Table from PostgreSQL
  - Cybertec: How to corrupt your PostgreSQL database
- AI tools:
  - Paid tier ChatGPT-4o/4.5 with Deep Research Option
  - Google Gemini Advanced 2.5 Pro with Deep Research Option



Questions?