credativ

UUIDv4 vs UUIDv7 in PostgreSQL 18

A Story of More Friendly UUID based Primary Keys

Josef Machytka <josef.machytka@credativ.de>

2025-12-05 - credativ Tech Talk

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, 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
- 33+ 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
- 3+ years of practical experience with different LLMs / AI / ML including architecture and principles
- From Czechia, living now 12 years in Berlin

- LinkedIn: linkedin.com/in/josef-machytka
- Medium: medium.com/@josef.machytka
- YouTube: youtube.com/@JosefMachytka

- GitHub: github.com/josmac69/conferences_slides
- ResearchGate: researchgate.net/profile/Josef-Machytka
- Academia.edu: academia.edu/JosefMachytka
- sessionize: sessionize.com/josefmachytka

All My Slides:

Recoded talks:

Topics We Will Cover

- What Are These UUIDs Anyway?
- Using UUIDs as PRIMARY KEYs in PostgreSQL
- Why UUIDv7 in PostgreSQL 18 is a big deal
- UUIDv4 vs UUIDv7 in table and index
- Index fragmentation and ordering
- Query performance

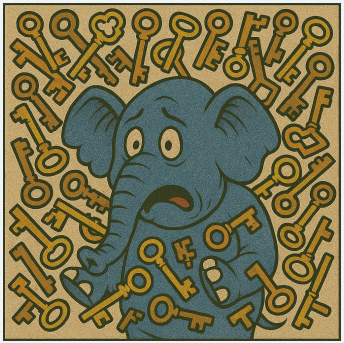


Image credit ChatGPT

What Are These UUIDs Anyway?

- UUID = Universally Unique Identifier
- 128-bit number -> i.e. 16 bytes integer value
- Represented as 32 hexadecimal digits
- Displayed in 5 groups separated by hyphens

- Existing versions of UUIDs:
  - UUIDv1: Time-based with MAC address
  - UUIDv3: Name-based using MD5 hash
  - UUIDv4: Randomly generated
  - UUIDv5: Name-based using SHA-1 hash
  - UUIDv6: Time-ordered (draft)
  - UUIDv7: Time-ordered with millisecond precision
  - UUIDv8: Custom implementations

Images from the article Understanding UUID: Purpose and Benefits of a Universal Unique Identifier

Where UUIDs Can Save Your Life

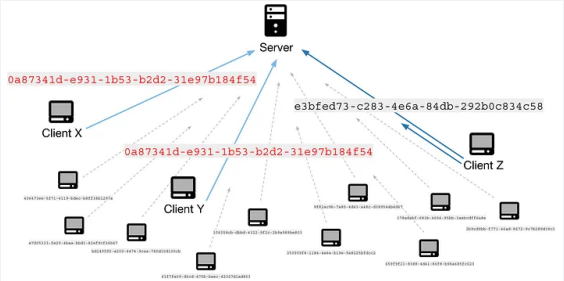
- If you play Minecraft, you are already using UUIDs every day
- Minecraft uses UUIDs to identify all objects in the game
- Every player, mob, item, block entity has a UUID
- Ensures uniqueness across servers and sessions
- Command /uuid in the command mode shows UUIDs of entities

Image from the article 9minecraft: UUID Command Mod

Where UUIDs Can Really Save You

- Areas where UUIDs are the only viable option:
  - Distributed databases
  - Microservices architectures
  - Client-side generated IDs
  - Merging datasets from different sources

- UUIDs help avoid collisions
- Ensure uniqueness across systems



Examples of UCA from Generating UUIDs at scale on the Web

PRIMARY KEYs in PostgreSQL

- PRIMARY KEY enforces uniqueness and NOT NULL
- Implemented as UNIQUE index on the column(s)
- Impact performance and scalability
- Foreign keys reference PRIMARY KEYs for data integrity

- Common choices for PRIMARY KEY:
  - SERIAL / BIGSERIAL (auto-incrementing integers)
  - INTEGER / BIGINT GENERATED ALWAYS AS IDENTITY
  - UUID (Universally Unique Identifier)

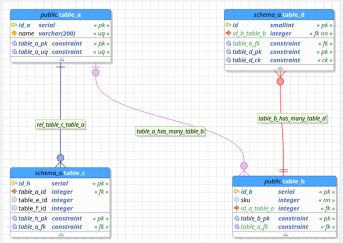


Image from PgModeler documentation

PRIMARY KEYs in PostgreSQL - Anti-Patterns

credativ

- Composite keys with many columns
  - Increases index size and complexity
  - B-tree entries store full key values
  - Large keys -> fewer entries per page
  - Deeper B-tree -> more IO lokups
  - More cache misses
  - Slows down joins and lookups
  - Foreign keys repeat the whole key
  - Partitioning key must include all key columns
  - Using text keys makes all problems worse

- Muttable primary keys - emails, usernames
  - Values can change over time
  - Requires updates to primary key column
  - Cascading updates to foreign keys

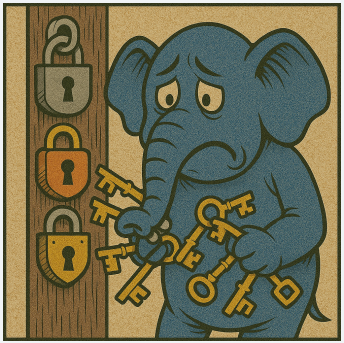
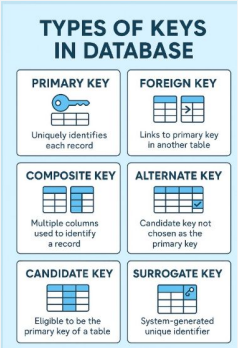


Image credit ChatGPT

PRIMARY KEYs in PostgreSQL

- INTEGER / BIGINT IDs
  - Simple and efficient
  - But can lead to collisions in distributed systems
  - And their range can be too small for large datasets
- UUIDs:
  - Provide much larger address space
  - Avoid collisions in distributed systems
  - Can be generated on the client side
  - But UUIDs version 4 hurt the performance
  - Random UUIDv4 scatter inserts across the whole index
  - Are considered anti-pattern by some DBAs



Examples of UCA from linkedin post about SQL data keys

UUID Uniqueness

- UUID is 128 bits = 16 bytes integer
- Total possible UUIDs = 2^{128} options
- It is about 3.4 x 10^{38} unique values

- Probability of collision is extremely low
- UUIDv4 - 1 in a billion after 10^{14} UUIDs
- V4 generated originally for distributed systems
- No central authority needed for uniqueness

- Known issue: 03000200-0400-0500-0006-000700080009
- Default UUID of many Gigabyte motherboards

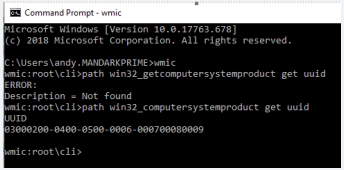


Image from the article Duplicate UUID - more common than expected

UUID Text Representation

- UUIDs are represented as text strings
- Standard format: 8-4-4-4-12 hexadecimal digits
- Example: 550e8400-e29b-41d4-a716-446655440000
- Structure: xxxxxxxx-xxxx-Vxxx-Wxxx-xxxxxxxxxxxx
- V - version (4 for UUIDv4, 7 for UUIDv7, etc.)
- W - variant nibble (top 2-3 bits - layout family)
- Hyphens improve readability

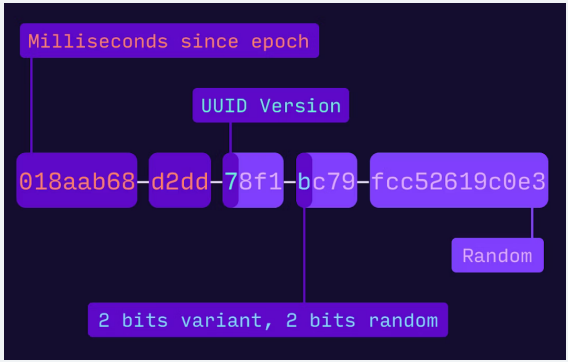
```
NIL UUID: 00000000-0000-0000-0000-000000000000
MAX UUID: ffffffff-ffff-ffff-ffff-ffffffffffff

-- UUIDv4 structure:
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           random_a                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          random_a             |  ver  |          random_b     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|var|                           random_c                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           random_c                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Image from the article RFC 9562 - Universally Unique IDentifiers

UUID version 7

- Includes 12-bit millisecond precision timestamp
- Still globally unique across systems
- Even better suited for distributed systems
- Improves index locality, reduces B-tree page splits

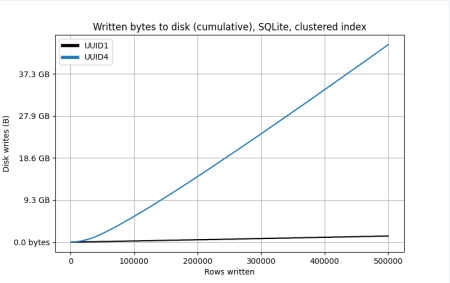
- But it also creates some security leakage
- Exposes timestamp of creating record
- Exposes time difference between records



Images from the article
Goodbye integers. Hello UUIDv7!

How Databases Store UUIDs

- PostgreSQL stores UUIDs as 16-byte binary value
- Some databases store UUIDs as 36 character text
  - Elasticsearch, CouchDB, BigQuery
- Other use binary/raw/blob(16) but no UUID type
  - MySQL, SQLite, Db2, Snowflake, TiDB, Oracle
- Other have native 16-byte UUID type
  - SQL Server, Cassandra, MongoDB, CockroachDB



Images from the article UUIDs Are Bad for Database Index Performance, enter UUID7

Native PostgreSQL UUID support

credativ

- PostgreSQL stores it as 16-byte binary value
- In source code src/include/utils/uuid.h
- struct pg_uuid_t unsigned char data[16]
- C handles char as 1 byte of opaque data
- opaque = no specific type interpretation

- uuidv4() - alias for gen_random_uuid() (v4, random)
- uuidv7() - generates time-ordered UUIDv7
- uuid_extract_version(uuid) - returns version: 4, 7
- uuid_extract_timestamp(uuid) - extracts timestamp

- Fun With UUIDs (PGDay Lowlands 2025)

UUIDv4 vs UUIDv7 Testing Setup

credativ

- Two tables with UUIDv4 and UUIDv7 PRIMARY KEYs
- Column ”ord” is a BIGINT identity column for insertion order
- Insert 1 million rows into each table

```
-- UUIDv4 (completely random keys)
CREATE TABLE uuidv4_demo (
    id   uuid PRIMARY KEY DEFAULT uuidv4(),   -- alias of gen_random_uuid()
    ord  bigint GENERATED ALWAYS AS IDENTITY );

-- UUIDv7 (time-ordered keys)
CREATE TABLE uuidv7_demo (
    id   uuid PRIMARY KEY DEFAULT uuidv7(),
    ord  bigint GENERATED ALWAYS AS IDENTITY );

-- 1M rows with UUIDv4
INSERT INTO uuidv4_demo (id) SELECT uuidv4() FROM generate_series(1, 1000000);

-- 1M rows with UUIDv7
INSERT INTO uuidv7_demo (id) SELECT uuidv7() FROM generate_series(1, 1000000);

VACUUM ANALYZE uuidv4_demo;
VACUUM ANALYZE uuidv7_demo;
```

UUIDv4 order of rows

credativ

- Values of UUIDv4 are completely randomly distributed across the table

```
SELECT row_number() OVER () AS seq_in_uuid_order, id, ord, ctid
FROM uuidv4_demo ORDER BY id LIMIT 20;
```

seq_in_uuid_order	id	ord	ctid
1	00000abf-cc8e-4cb2-a91a-701a3c96bd36	673969	(4292,125)
2	00001827-16fe-4aee-9bce-d30ca49ceb1d	477118	(3038,152)
3	00001a84-6d30-492f-866d-72c3b4e1edff	815025	(5191,38)
4	00002759-21d1-4889-9874-4a0099c75286	879671	(5602,157)
5	00002b44-b1b5-473f-b63f-7554fa88018d	729197	(4644,89)
6	00002ceb-5332-44f4-a83b-fb8e9ba73599	797950	(5082,76)
7	000040e2-f6ac-4b5e-870a-63ab04a5fa39	160314	(1021,17)
8	000053d7-8450-4255-b320-fee8d6246c5b	369644	(2354,66)
9	00009c78-6eac-4210-baa9-45b835749838	463430	(2951,123)
10	0000a118-f98e-4e4a-acb3-392006bcabb8	96325	(613,84)
11	0000be99-344b-4529-aa4c-579104439b38	454804	(2896,132)
12	00010300-fcc1-4ec4-ae16-110f93023068	52423	(333,142)
13	00010c33-a4c9-4612-ba9a-6c5612fe44e6	82935	(528,39)
14	00011fa2-32ce-4ee0-904a-13991d451934	988370	(6295,55)
15	00012920-38c7-4371-bd15-72e2996af84d	960556	(6118,30)
16	00014240-7228-4998-87c1-e8b23b01194a	66048	(420,108)
17	00014423-15fc-42ca-89bd-1d0acf3e5ad2	250698	(1596,126)
18	000160b9-a1d8-4ef0-8979-8640025c0406	106463	(678,17)
19	0001711a-9656-4628-9d0c-1fb40620ba41	920459	(5862,125)
20	000181d5-ee13-42c7-a9e7-0f2c52faeadb	513817	(3272,113)

UUIDv7 order of rows

credativ

- Values of UUIDv7 are ordered sequentially in the table

seq_in_uuid_order	id	ord	ctid
1	019ad94d-0127-7aba-b9f6-18620afdea4a	1	(0,1)
2	019ad94d-0131-72b9-823e-89e41d1fad73	2	(0,2)
3	019ad94d-0131-7384-b03d-8820be60f88e	3	(0,3)
4	019ad94d-0131-738b-b3c0-3f91a0b223a8	4	(0,4)
5	019ad94d-0131-7391-ab84-a719ca98accf	5	(0,5)
6	019ad94d-0131-7396-b41d-7f9f27a179c4	6	(0,6)
7	019ad94d-0131-739b-bdb3-4659aeaafbdd	7	(0,7)
8	019ad94d-0131-73a0-b271-7dba06512231	8	(0,8)
9	019ad94d-0131-73a5-8911-5ec5d446c8a9	9	(0,9)
10	019ad94d-0131-73aa-a4a3-0e5c14f09374	10	(0,10)
11	019ad94d-0131-73af-ac4b-3710e221390e	11	(0,11)
12	019ad94d-0131-73b4-85d6-ed575d11e9cf	12	(0,12)
13	019ad94d-0131-73b9-b802-d5695f5bf781	13	(0,13)
14	019ad94d-0131-73be-bcb0-b0775dab6dd4	14	(0,14)
15	019ad94d-0131-73c3-9ec8-c7400b5c8983	15	(0,15)
16	019ad94d-0131-73c8-b067-435258087b3a	16	(0,16)
17	019ad94d-0131-73cd-a03f-a28092604fb1	17	(0,17)
18	019ad94d-0131-73d3-b4d5-02516d5667b5	18	(0,18)
19	019ad94d-0131-73d8-9c41-86fa79f74673	19	(0,19)
20	019ad94d-0131-73dd-b9f1-dcd07598c35d	20	(0,20)

Statistics of UUIDv4 vs UUIDv7 Columns

credativ

- VACUUM ANALYZE collects statistics about table columns
- Correlation shows how well physical order matches index order

```
SELECT
    tablename,
    attname,
    correlation
FROM pg_stats
WHERE tablename IN ('uuidv4_demo', 'uuidv7_demo')
AND attname = 'id'
ORDER BY tablename, attname;

   tablename | attname | correlation
-------------+---------+--------------
 uuidv4_demo |      id | -0.0024808696  -- no correlation
 uuidv7_demo |      id |             1  -- perfect correlation
```

Primary Key Indexes Analysis

- We look at sizes, number of leaf pages, density, fragmentation

```
SELECT 'uuidv4_demo_pkey' AS index_name, (pgstatindex('uuidv4_demo_pkey')).*;
        index_name | uuidv4_demo_pkey
           version | 4
        tree_level | 2
        index_size | 40026112  -- index 26% bigger than uuidv7
     root_block_no | 295
    internal_pages | 24
        leaf_pages | 4861      -- more leaf pages 4861 vs 3832
       empty_pages | 0
     deleted_pages | 0
  avg_leaf_density | 71        -- lower density 71%
leaf_fragmentation | 49.99     -- very high fragmentation 50%

SELECT 'uuidv7_demo_pkey' AS index_name, (pgstatindex('uuidv7_demo_pkey')).*;
        index_name | uuidv7_demo_pkey
           version | 4
        tree_level | 2
        index_size | 31563776
     root_block_no | 295
    internal_pages | 20
        leaf_pages | 3832
       empty_pages | 0
     deleted_pages | 0
  avg_leaf_density | 89.98     -- default fillfactor 90%
leaf_fragmentation | 0.00
```

Deeper Indexes Analysis

credativ

- Statistics about leaf pages of the primary key index

```
WITH leaf AS (
    SELECT *
    FROM bt_multi_page_stats('uuidv4_demo_pkey', 1, -1) -- from block 1 to end
    WHERE type = 'l'
)
SELECT
    count(*) AS leaf_pages,
    min(blkno) AS first_leaf_blk,
    max(blkno) AS last_leaf_blk,
    max(blkno) - min(blkno) + 1 AS leaf_span,
    round( count(*)::numeric / (max(blkno) - min(blkno) + 1), 3) AS leaf_density_by_span,
    min(live_items) AS min_tuples_per_page,
    max(live_items) AS max_tuples_per_page,
    avg(live_items)::numeric(10,2) AS avg_tuples_per_page,
    sum(CASE WHEN btpo_next = blkno + 1 THEN 1 ELSE 0 END) AS contiguous_links,
    sum(CASE WHEN btpo_next <> 0 AND btpo_next <> blkno + 1 THEN 1 ELSE 0 END) AS non_contiguous_links
FROM leaf;
```

Deeper Indexes Analysis

credativ

- Statistics about leaf pages of the primary key index

```
-- uuidv4_demo_pkey
         leaf_pages | 4861
      first_leaf_blk | 1
       last_leaf_blk | 4885
           leaf_span | 4885
leaf_density_by_span | 0.995
 min_tuples_per_page | 146      -- these pages are half empty
 max_tuples_per_page | 291      -- some pages are almost full, above fillfactor 90%
 avg_tuples_per_page | 206.72
    contiguous_links | 0        -- no pages following each other
non_contiguous_links | 4860     -- almost every link is non-contiguous

-- uuidv7_demo_pkey
         leaf_pages | 3832
      first_leaf_blk | 1
       last_leaf_blk | 3852
           leaf_span | 3852
leaf_density_by_span | 0.995
 min_tuples_per_page | 109      -- the very last page is not full
 max_tuples_per_page | 262      -- number of tuples in all other pages
 avg_tuples_per_page | 261.96   -- max and avg almost equal
    contiguous_links | 3812     -- almost all pages follow each other
non_contiguous_links | 19
```

Chain of Leaf Pages - UUIDv4

- Let's visualize the chain of leaf pages to see fragmentation better

```
SELECT
    blkno, case when type='l' then 'leaf' when type='i' then 'internal' end as type,
    btpo_prev, btpo_next, live_items
FROM bt_multi_page_stats('uuidv4_demo_pkey', 1::bigint, -1::bigint) LIMIT 20;
```

blkno	type	btpo_prev	btpo_next	live_items
1	leaf	0	3744	209
2	leaf	4149	4373	164
3	internal	0	3285	248
4	leaf	3656	4331	184
5	leaf	4569	2441	291
6	leaf	3507	2577	275
7	leaf	2537	3167	230
8	leaf	2529	4536	160
9	leaf	2554	3125	215
10	leaf	3189	3502	217
11	leaf	4798	3238	225
12	leaf	3497	3045	238
13	leaf	3694	4741	148
14	leaf	4797	4613	159
15	leaf	3985	2253	276
16	leaf	3933	2606	265
17	leaf	3553	3902	204
18	leaf	4617	4163	183
19	leaf	4647	4430	159
20	leaf	4379	4228	181

Chain of Leaf Pages – UUIDv7

credativ

- UUIDv7 leaf pages nicely follow each other

```
SELECT
    blkno, case when type='l' then 'leaf' when type='i' then 'internal' end as type,
    btpo_prev, btpo_next, live_items
FROM bt_multi_page_stats('uuidv7_demo_pkey', 1::bigint, -1::bigint) LIMIT 20;
```

blkno	type	btpo_prev	btpo_next	live_items
1	leaf	0	2	262
2	leaf	1	4	262
3	internal	0	294	205
4	leaf	2	5	262
5	leaf	4	6	262
6	leaf	5	7	262
7	leaf	6	8	262
8	leaf	7	9	262
9	leaf	8	10	262
10	leaf	9	11	262
11	leaf	10	12	262
12	leaf	11	13	262
13	leaf	12	14	262
14	leaf	13	15	262
15	leaf	14	16	262
16	leaf	15	17	262
17	leaf	16	18	262
18	leaf	17	19	262
19	leaf	18	20	262
20	leaf	19	21	262

Histogram of Leaf Pages Tuple Counts

credativ

- Statistics about leaf pages of the primary key index

```
WITH leaf AS (
    SELECT live_items
    FROM bt_multi_page_stats('uuidv4_demo_pkey', 1, -1)
    WHERE type = 'l'
),
buckets AS (
    -- bucket lower bounds: 100, 110, ..., 290
    SELECT generate_series(100, 290, 10) AS bucket_min
)
SELECT
    b.bucket_min AS bucket_from,
    b.bucket_min + 9 AS bucket_to,
    COUNT(l.live_items) AS page_count
FROM buckets b
LEFT JOIN leaf l
    ON l.live_items BETWEEN b.bucket_min AND b.bucket_min + 9
GROUP BY b.bucket_min HAVING count(l.live_items) > 0
ORDER BY b.bucket_min;
```

Histogram of Leaf Pages Tuple Counts

- How many leaf pages have 100-109, 110-119, ..., 290-299 tuples

```
-- Result for UUIDv4
 bucket_from | bucket_to | page_count
-------------+-----------+------------
         140 |       149 |        159    -- half empty pages
         150 |       159 |        435
         160 |       169 |        388
         170 |       179 |        390
         180 |       189 |        427
         190 |       199 |        466
         200 |       209 |        430
         210 |       219 |        387
         220 |       229 |        416
         230 |       239 |        293
         240 |       249 |        296
         250 |       259 |        228
         260 |       269 |        214    -- fillfactor 90%
         270 |       279 |        171
         280 |       289 |        140
         290 |       299 |         21

-- Result for UUIDv7
 bucket_from | bucket_to | page_count
-------------+-----------+------------
         100 |       109 |          1
         260 |       269 |       3831    -- all pages have 262 tuples
```

UUIDv7 Exposes Insertion Time

credativ

- PostgreSQL 18 even has function uuid_extract_timestamp()
- Extracts timestamp from UUIDv7 value
- Can be used to analyze insertion patterns

```
SELECT
    id,
    uuid_extract_timestamp(id) AS created_at_from_uuid
FROM uuidv7_demo
ORDER BY ord
LIMIT 5;
Sample results:

                  id                  |       created_at_from_uuid
--------------------------------------+----------------------------
 019ad94d-0127-7aba-b9f6-18620afdea4a | 2025-12-01 09:44:53.799+00
 019ad94d-0131-72b9-823e-89e41d1fad73 | 2025-12-01 09:44:53.809+00
 019ad94d-0131-7384-b03d-8820be60f88e | 2025-12-01 09:44:53.809+00
 019ad94d-0131-738b-b3c0-3f91a0b223a8 | 2025-12-01 09:44:53.809+00
 019ad94d-0131-7391-ab84-a719ca98accf | 2025-12-01 09:44:53.809+00
```

EXPLAIN ANALYZE UUIDv4 vs UUIDv7

credativ

- Query plans show performance differences between UUIDv4 and UUIDv7

```
-- UUIDv4
EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM uuidv4_demo ORDER BY id;

Index Scan using uuidv4_demo_pkey on uuidv4_demo
        (cost=0.42..60024.31 rows=1000000 width=24) (actual time=0.031..301.163 rows=1000000.00 loops=1)
  Index Searches: 1
  Buffers: shared hit=1004700 read=30
Planning Time: 0.109 ms
Execution Time: 318.005 ms


-- UUIDv7
EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM uuidv7_demo ORDER BY id;

Index Scan using uuidv7_demo_pkey on uuidv7_demo
        (cost=0.42..36785.43 rows=1000000 width=24) (actual time=0.013..96.177 rows=1000000.00 loops=1)
  Index Searches: 1
  Buffers: shared hit=2821 read=7383
Planning Time: 0.040 ms
Execution Time: 113.305 ms
```

UUIDv4 vs UUIDv7 Benchmarking

credativ

- For benchmarking I compared time taken to insert 50 million rows
- First into empty table, then into table with 50 million existing rows
- These tests showed huge differences in performance

```
INSERT INTO uuidv4_demo (id) SELECT uuidv4() FROM generate_series(1, 50000000);
INSERT INTO uuidv7_demo (id) SELECT uuidv7() FROM generate_series(1, 50000000);

-- UUID v4                                  -- UUID v7
                                Empty table
Insert time: 1239839.702 ms (20:39.840)     Insert time: 106343.314 ms (01:46.343)
Table size: 2489 MB                         Table size: 2489 MB
Index size: 1981 MB                         Index size: 1504 MB

                               Table with 50M rows
Insert time: 2776880.790 ms (46:16.881)     Insert time: 100354.087 ms (01:40.354)
Table size: 4978 MB                         Table size: 4978 MB
Index size: 3956 MB                         Index size: 3008 MB
```

Summary

credativ

- UUIDs provide an enormous identifier space and global uniqueness
- Traditional UUIDv4 are random, cause severe index fragmentation
- Lead to larger indexes, very poor insertion performance

- UUIDv7 is time-ordered, improves index locality
- Reduces index size, page splits, fragmentation
- Greatly improves insertion performance
- PostgreSQL 18 has native uuidv7() function

- But UUIDv7 exposes insertion timestamps
- May not be suitable for all use cases

Thank you for your attention!

credativ

All my slides

Recorded talks