How PostgreSQL Stores JSONB Data in TOAST Tables

Josef Machytka
5 min read · 4 days ago

During my project testing various use cases and internals related to JSONB data in PostgreSQL, I also explored how PostgreSQL handles JSONB storage. For these tests, I used GitHub historical events downloaded from gharchive.org. I downloaded one week of data from 2023.01.01 to 2023.01.07, which was 17.4 million rows. These historical events are stored as JSON documents. Below is an example of one such record:


```
{ "id": "26167585827",
"repo": {"id": 581592468,
        "url": "https://api.github.com/repos/tiwabs/tiwabs_audio_door_tool",
        "name": "tiwabs/tiwabs_audio_door_tool"
        },
"type": "PushEvent",
"actor": {"id": 48737497,
          "url": "https://api.github.com/users/tiwabs",
          "login": "tiwabs",
          "avatar_url": "https://avatars.githubusercontent.com/u/48737497?",
          "gravatar_id": "",
          "display_login": "tiwabs"
          },
"public": true,
"payload": {"ref": "refs/heads/master",
            "head": "3ca247941f269bcedeb17e5b12e9b3b74b1c4da2",
            "size": 1,
            "before": "0dd5471667b12084b8fc88b1bca299780382d50a",
            "commits":[ { "sha": "3ca247941f269bcedeb17e5b12e9b3b74b1c4da2",
                          "url": "https://api.github.com/repos/tiwabs/tiwabs_au
                          "author": {"name": "Tiwabs",
                                     "email": "mrskielz@gmail.com"
```

```
                                    },
                                "message": "fix(export): export nametable if export s
                                "distinct": true}
                        ],
            "push_id": 12149772587,
            "distinct_size": 1
            },
"created_at": "2023-01-01T13:39:55Z"
}
```


I specifically chose such non-trivial JSON documents because they closely resemble real-world examples seen in client projects, both in complexity and size.

The historical GitHub records varied significantly in size. Let's recall that JSONB is stored as a binary object with a tree structure of nodes, and PostgreSQL allows a maximum size of 1/4 GB — 1 byte (i.e., 256 MB — 1 byte). This limit stems from PostgreSQL's broader 1 GB memory allocation restriction, with a conservative threshold for JSONB to account for potential character escaping.

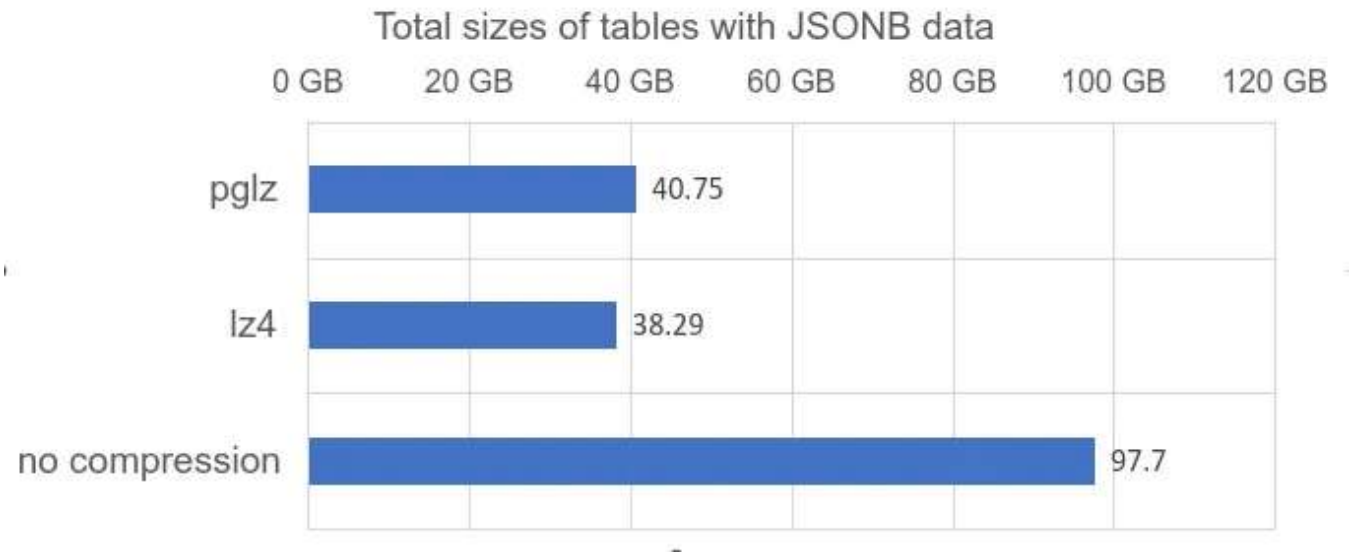
Table Structures for Testing

I used a straightforward table structure, mimicking what I see by our clients, and created three tables for testing different compression methods. The first table used the older `pglz` compression, the second leveraged the newer `lz4` compression, and the third stored data externally without compression:

```
CREATE TABLE public.github_events_pglz (
    id bigserial NOT NULL,
    jsonb_data jsonb compression pglz NULL,
CONSTRAINT github_events_pglz_pkey PRIMARY KEY (id) );

CREATE TABLE public.github_events_lz4 (
    id bigserial NOT NULL,
    jsonb_data jsonb compression lz4 NULL,
CONSTRAINT github_events_lz4_pkey PRIMARY KEY (id) );

CREATE TABLE public.github_events_nocompression (
    id bigserial NOT NULL,
    jsonb_data jsonb storage external NULL,
CONSTRAINT github_events_nocompression_pkey PRIMARY KEY (id) );
```

Storage Efficiency

The first graph in my analysis highlights the total sizes of these tables after loading identical data into each. The new `lz4` compression proved to be the most efficient.



I already discussed the performance characteristics of these compression types in my article on the NetApp-credativ blog, where you can find results of performance tests of sequential scans over the whole tables. These shows that `lz4` compression is the quickest one.

Understanding TOAST

Data objects exceeding the 8 KB data page limit are in PostgreSQL stored using TOAST (The Oversized Attribute Storage Technique). Any table with `TEXT` columns automatically gets a TOAST table, identified by its `relid` in `pg_class.reltoastrelid`. However, whether data ends up in the TOAST table depends both on its original size and its size after compression.

For `pglz` and `lz4` compression, the source file `heaptoast.h` in PostgreSQL repository explains the behavior:

> *"If a tuple is larger than TOAST_TUPLE_THRESHOLD, we will try to toast it down to no more than TOAST_TUPLE_TARGET bytes through compressing compressible fields and moving data out-of-line."*

Both thresholds are initialized to 2,000 bytes, calculated to allow four tuples on an 8 KB data page. When we opt for "external" storage without compression, all JSON documents larger than 2,000 bytes are stored in the TOAST table without compression.

Compression Scenarios

These thresholds mean that in real tables, data stored directly in the tuple in the main table may or may not be compressed, depending on its size after compression.

To check if columns are compressed and which algorithm is used, PostgreSQL give us `pg_column_compression` function. To see what is the size of data in the column (including compression) we can use `pg_column_size` and in PostgreSQL 17 we have a new function, `pg_column_toast_chunk_id`, which indicates whether a column's value is stored in the TOAST table.

Here is example select:

```
SELECT
    pg_column_compression(jsonb_data) as column_compression,
    (pg_column_toast_chunk_id(jsonb_data) is not null) as column_toasted,
    min(pg_column_size(jsonb_data)) as min_column_size,
    max(pg_column_size(jsonb_data)) as max_column_size,
    count(*) as count
FROM github_events_pglz
GROUP BY 1,2
ORDER BY 1,2;
```

For the `lz4` table, the results were:

```
 column_compression | column_toasted | min_column_size | max_column_size |  cou
--------------------+----------------+-----------------+-----------------+-----
                    | f              |             440 |            2004 | 1267
 lz4                | f              |             679 |            2004 |   72
 lz4                | t              |            2001 |          269140 |  407
(3 rows)
```

There are not many details about the the function *pg_column_size* neither in documentation nor in source code. But it looks like threshold value 2004 as column size in case of toasted value includes also "*varlena header*" of 4 bytes as is shown in PostgreSQL source code.

Anyway, cases without compression are JSON data which size was smaller or equal 2000 bytes, and these were stored directly in the tuple, without compression. Flag "*column_toasted*" is false for these records and *column_compression* is NULL.

We also see records which are compressed but not toasted, stored directly in the tuple. Those are JSON documents which exceeded 2000 bytes, PostgreSQL compressed them, but because their size after compression was smaller than second threshold, which is also 2000 bytes, these compressed values were therefore also stored directly in the tuple.

Third case are JSONB data which after compression exceeded 2000 bytes and therefore were moved into TOAST table and original tuple contains only 18 bytes long TOAST pointer, corresponding with chunk_id in the TOAST table, as is explained in PostgreSQL documentation.

Insights into TOAST Tables

We also should check, how are these compressed records further handled in TOAST table. Because this is after all just another table, and total size of tuple in it is also limited to 8 KB.

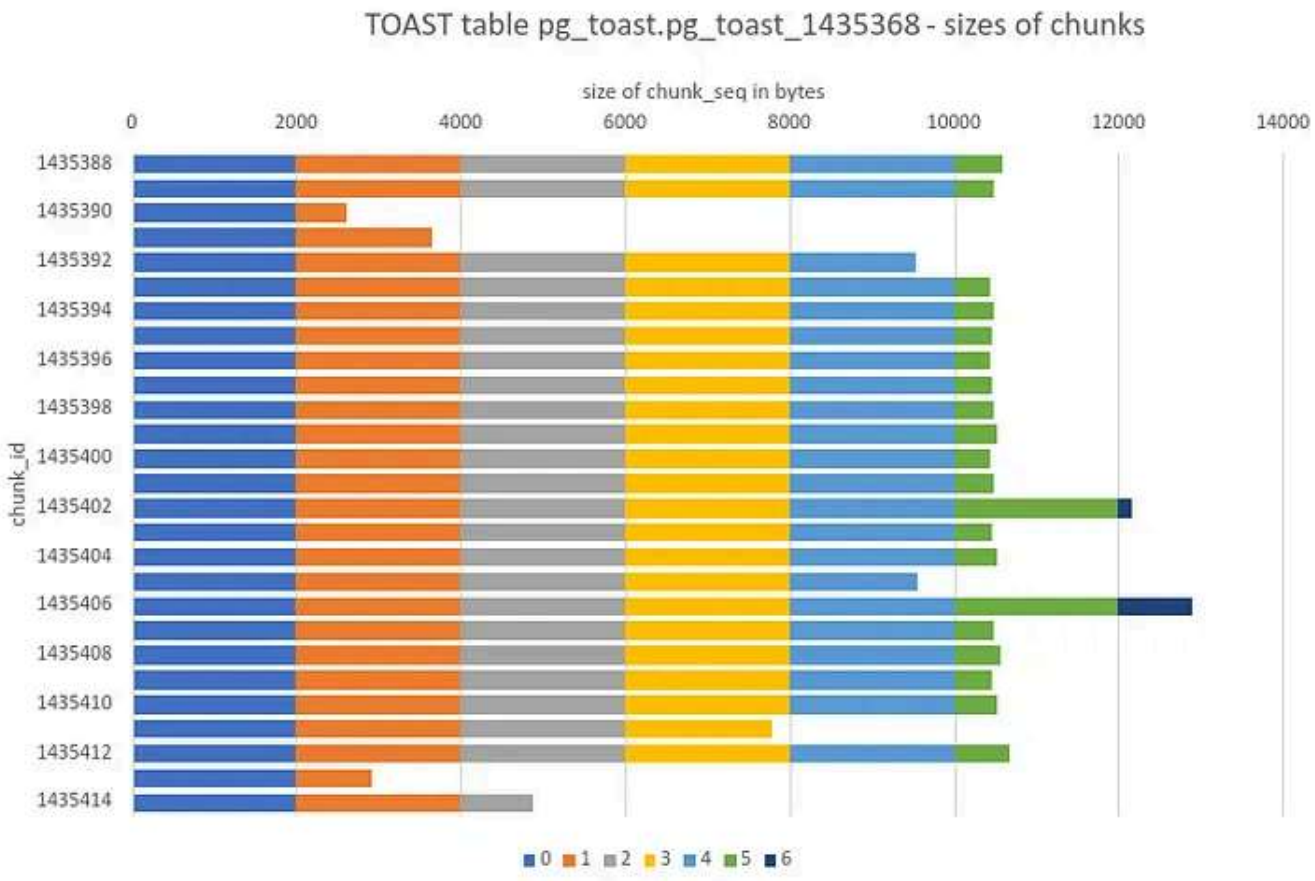
The TOAST table structure is straightforward:

```
postgres=# \dS+ pg_toast.pg_toast_16388

TOAST table "pg_toast.pg_toast_16388"
   Column   |  Type   | Storage
------------+---------+---------
 chunk_id   | oid     | plain
 chunk_seq  | integer | plain
 chunk_data | bytea   | plain

Owning table: "public.github_events_load_20240421"
Indexes:
    "pg_toast_16388_index" PRIMARY KEY, btree (chunk_id, chunk_seq)
Access method: heap
```

To overcome the 8 KB limit, PostgreSQL divides toasted values into 2,000-byte chunks. All parts of one toasted value have the same `chunk_id` , and have sequential `chunk_seq` numbers starting from 0. Each chunk is stored as a separate record, with

the remaining part stored in a smaller final chunk. This mechanism ensures efficient storage and retrieval of large values.

Following graph shows it in colors:



Conclusion

TOAST tables offer a simple and efficient mechanism for storing values exceeding PostgreSQL's 8 KB page size limit. While not universally optimal, their implementation is robust for many use cases.

Image created by the author using DeepDreamGenerator

Postgresql Json Jsonb Toast

Edit profile

Written by Josef Machytka

58 Followers · 15 Following

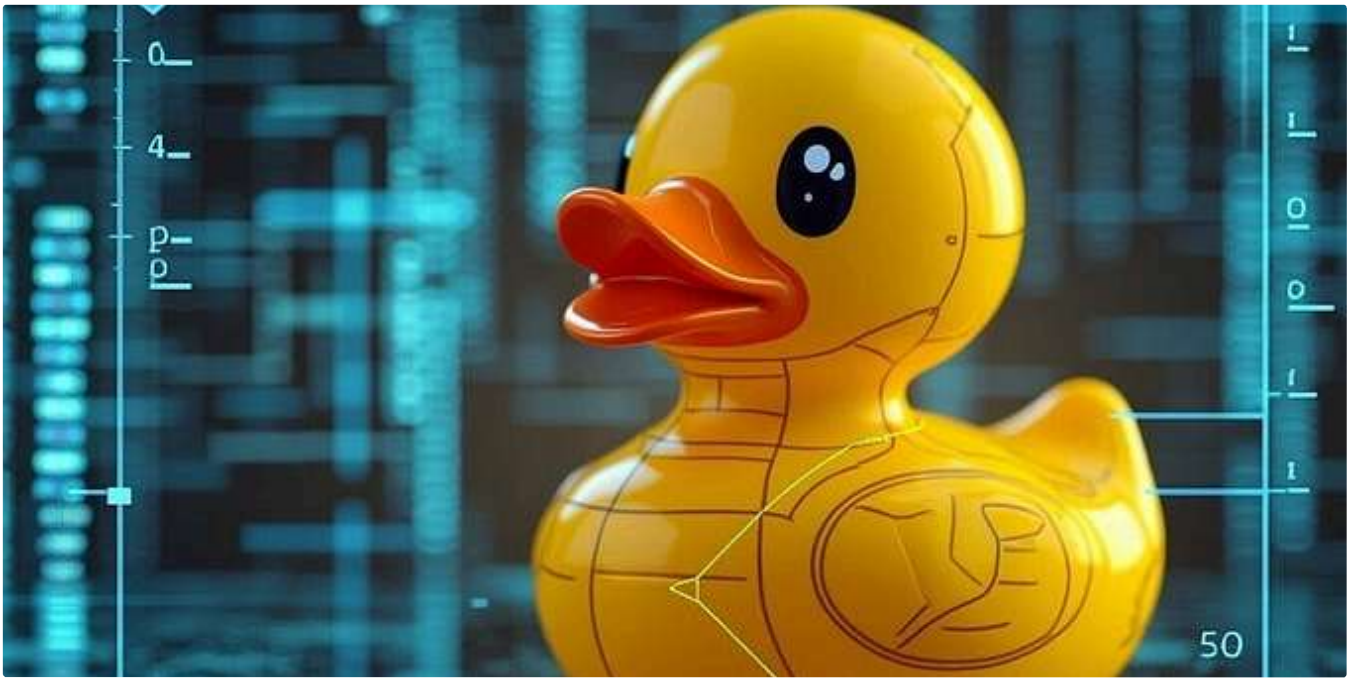
I work as PostgreSQL specialist & database reliability engineer at NetApp Deutschland, Open Source Services division.

No responses yet

What are your thoughts?

Respond

More from Josef Machytka

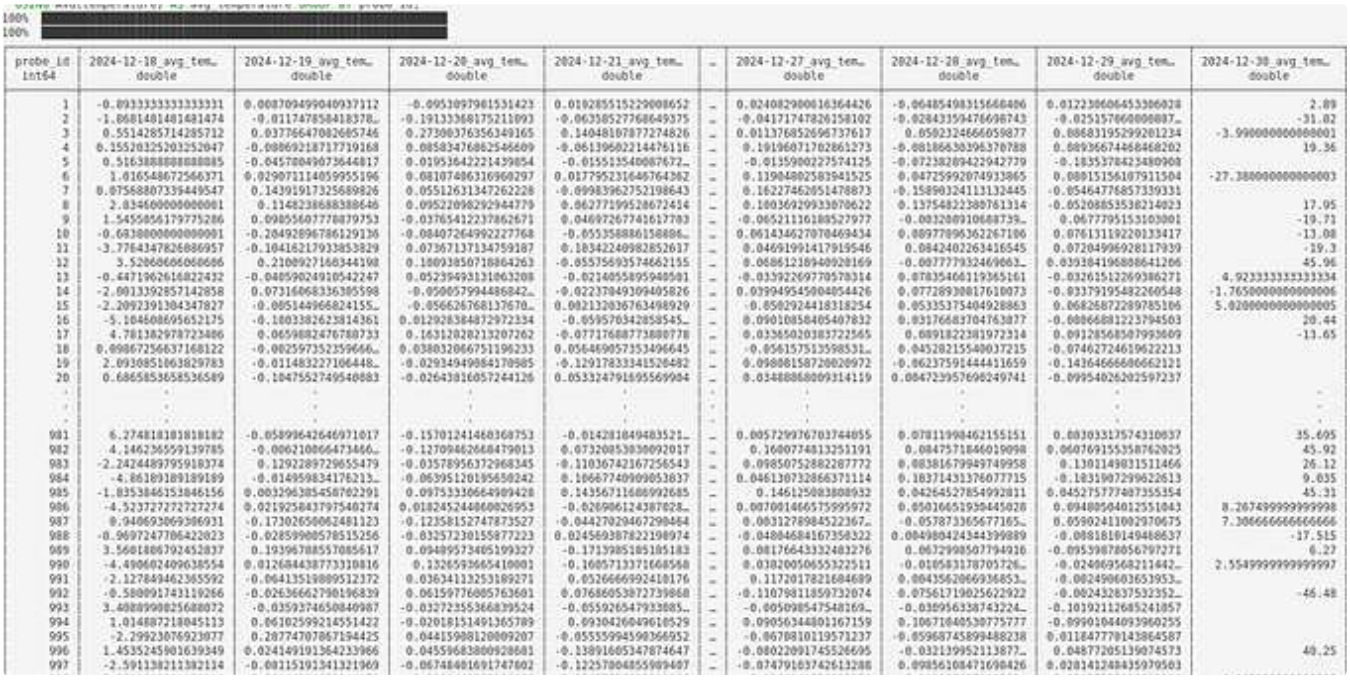


Josef Machytka

DuckDB Database File as a New Standard for Sharing Data?

This is not my original idea; I came across it in an excellent article titled “DuckDB Beyond the Hype” by Alireza Sadeghi. However, it...

Dec 30, 2024 17 2



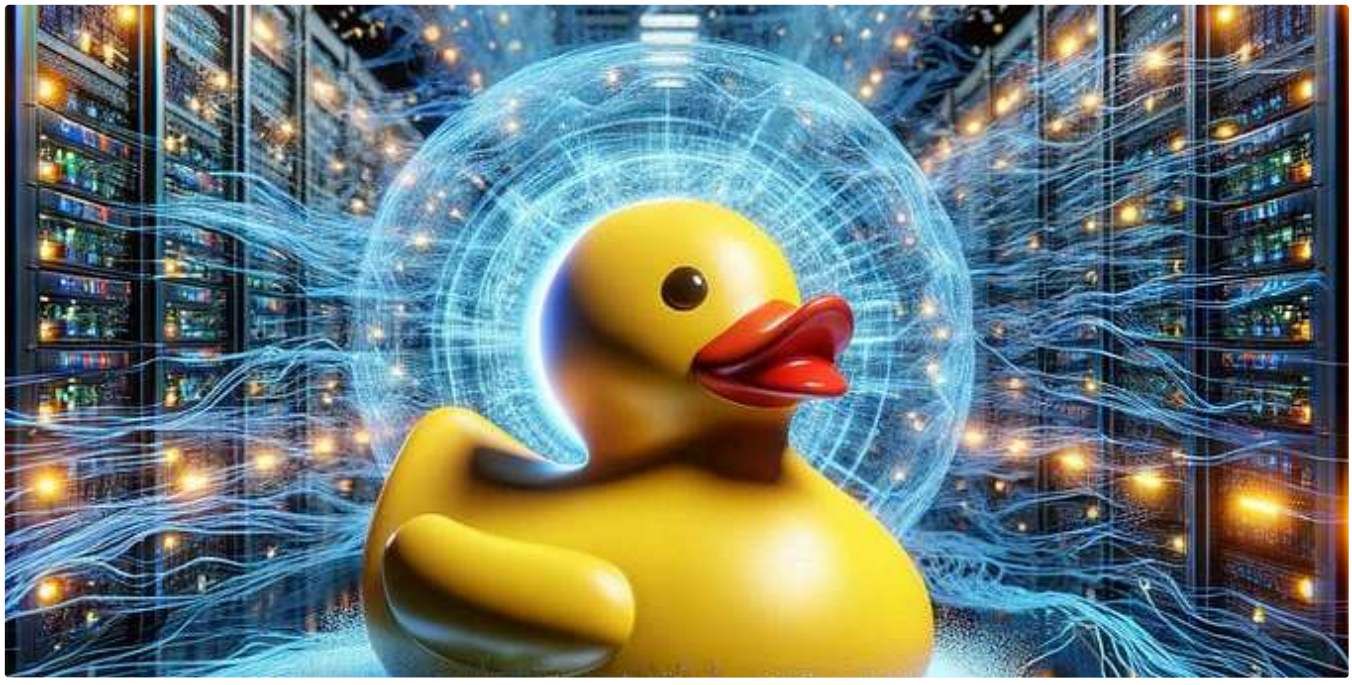
probe_id int64	2024-12-18_avg_tem… double	2024-12-19_avg_tem… double	2024-12-20_avg_tem… double	2024-12-21_avg_tem… double	…	2024-12-27_avg_tem… double	2024-12-28_avg_tem… double	2024-12-29_avg_tem… double	2024-12-30_avg_tem… double
1	-0.0933333333333331	0.008709499040937112	-0.09530979615314230	0.010285515229008652	…	0.024082900616364426	-0.06485498315668406	0.012230606453306028	2.89
2	-1.8681481481481474	-0.011747858418378…	-0.19133368175211093	-0.06350527768649375	…	-0.041717478261581020	-0.028433594766987430	-0.025157060000087…	-31.82
3	0.5514285714285712	0.037766470826857460	0.27300376356349165	0.140481078772748260	…	0.011376852696737617	0.05923246666598770	0.088831952992012340	-3.990000000000001
4	0.15520325203252047	-0.008692187717719168	0.08583476862546609	-0.061396022144761160	…	0.191960717022861273	-0.081866303963707880	0.089366744868468202	19.36
5	0.5163888888888885	-0.045780490736444817	0.01953642221439854	-0.015513540087672…	…	-0.013590022757541250	-0.072382894229427790	-0.183537842480908	
6	1.0165486725663710	0.029071114059955196	0.08107486316960297	0.017795231646764362	…	0.119048025835941525	0.047259920749133865	0.088151356107911504	-27.380000000000003
7	0.07568807339449547	0.14391917325689826	0.05512613147262228	-0.09983962752198643	…	0.162274620514788730	-0.158903241131324450	-0.054647768573393310	
8	2.034600000000001	0.11482386883886460	0.09522098292944779	0.062771995286724140	…	0.180369299330706220	0.137548223807613140	-0.052088535382149230	17.05
9	1.5455856179775286	0.098556077780879530	-0.03765412237862671	0.046972677416177030	…	-0.065211161885279770	-0.003208910688739…	0.06777951153103001	-19.71
10	-0.6838000000000001	-0.284928967861291360	-0.08407264992227760	-0.055358886158886…	…	0.061434627070469434	0.089778963622671860	0.076131198220133417	-13.08
11	-3.7764347826086957	-0.104162179338538290	0.07367137134759187	0.183422409828526170	…	0.046919914179195460	0.08424022634165450	0.072049969281179390	-19.3
12	3.520606060606606	0.210092716834419080	0.18093850718864263	-0.055756935746621550	…	0.068612189409281690	-0.007777932469063…	0.039384196880641266	45.96
13	-0.4471962616822432	-0.040590249105422470	0.05239495131063208	-0.021405589594085010	…	-0.013922697705783140	0.078354661193651610	-0.032615122693862710	4.923333333333334
14	-2.001339285714285 8	0.073160683363855980	-0.05005799448684220…	-0.022370491099405826	…	0.039949545004854426	0.077269308176100730	-0.033791954822260540	-1.7650000000000006
15	-2.209239313043478 27	-0.005144966842415 5…	-0.05662676811376700…	0.082112036763498929	…	-0.05029244183182540	0.053353754049288630	0.068268722897851060	5.020000000000005
16	-5.104608695652175	-0.180338262381436 1	0.01292838487297233 4	-0.059576342858545…	…	0.090108584054078320	0.031766837047638770	-0.008668812233794503	20.44
17	4.781382978723486	0.065988247678078330	0.16312028213207262	-0.077176887738887780	…	0.033650203837225650	0.089182238197231480	0.091285685079936090	-13.65
18	0.09867256637168122	-0.002597352359666…	0.03803206675119623 3	0.056469057353496645	…	-0.056157513598531…	0.045282155400372150	-0.074627246169622130	
19	2.093085106382978 3	-0.011483227106448…	-0.02934949084170585	-0.129178333415284820	…	0.096081587200209720	-0.062375914444116590	-0.143646666860662120	
20	0.6865853658536589	-0.104755274954088 3	-0.02643816057244126	0.053324791695569904	…	0.034888680093141190	0.084723957690249741	-0.099540262025972370	
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…
981	6.274818181818182	-0.058996426469710170	-0.15701241468368753	-0.014281849483521…	…	0.005729976703744055	0.078119984621551510	0.083033137574310370	35.695
982	4.146236559139785	-0.006210866473466…	-0.12709462668479013	0.073208530309021070	…	0.160077481325119 1	0.084757184601909 8	0.060769155358762025	45.92
983	-2.242448979591837 4	0.129228972965547 9	-0.03578956372968345	-0.110367421672565 43	…	0.098507528822877720	0.083816799497499 58	0.130114983115114660	26.12
984	-4.861891891891899	-0.014959834176213…	-0.06395120195650242	0.106677409009053870	…	0.046130732866371114	0.183714313760777150	-0.183190729962261 3	9.035
985	-1.835384615384615 6	0.003296385458702291	0.09753330664989428	0.143567116869926850	…	0.146125083808932	0.042645278549928110	0.045275777407355354	45.31
986	-4.523727272727274	0.021925843797540274	0.01824524486002695 3	-0.026906124387028…	…	0.007001466575995972	0.050166519904450280	0.094805040125510430	8.267499999999998
987	0.9406930693069310	-0.173026500624811 23	-0.12358152747873527	-0.044270294672904640	…	0.003127898452236 7…	-0.057873365677165…	0.059024110029706750	7.306666666666666
988	-0.9697247706422023	-0.028599080578515256	-0.03257230155877223	0.024569387822198974	…	-0.048046841673583220	0.004980424344399889	-0.088181014946863 7	-17.515
989	3.5681886792452837	0.193967885570856 17	0.09489573405199327	-0.17139851851851 83	…	0.081766433324832760	0.067299850779491 60	-0.095398780567972710	6.27
990	-4.490602409638554	0.012684438773310816	0.13265936665410001	0.160571337166856 8	…	0.038200506553225110	-0.010583178705726…	-0.024069568211442…	2.554999999999997
991	-2.127849462365592	-0.064135198895123720	0.03634113253189271	0.052666699241017 6	…	0.117201782168468 9	0.004356206693685 3…	-0.082490603653953…	
992	-0.5800917431192660	-0.026366627901968390	0.06159776805763601	0.076860538727339868	…	-0.110798118597320740	0.075617190256229220	-0.002432837532352…	-46.48
993	3.4088999082568807 2	-0.035937465084098 70	-0.03272355366839524	-0.055926547933085…	…	-0.005098547548169…	-0.038956338743224…	-0.101921126852418570	
994	1.0146872180451130	0.061025992145514220	-0.02018151491365789	0.093042604961052 9	…	0.090563448011671590	0.106710405307757770	-0.099010440933960255	
995	-2.299230769230770	0.287747078671944250	0.04415988120009207	-0.055559945903669520	…	-0.067081011957123700	-0.059687458994882380	0.011847770143864587	
996	1.4535245901639349	0.024149191364233966	0.04559683800928681	-0.138916053478746470	…	-0.080220917455266950	-0.032139952113877…	0.048772051390745730	40.25
997	-2.591138211382114	-0.081151913413219690	-0.06748481691747602	-0.122578048559094070	…	-0.074791637426132880	0.098561084716984260	0.028141248435979503	

Josef Machytka

DuckDB Performance Problems with Inappropriate Pivoting Queries on Very Large Datasets

The DuckDB documentation clearly states that this tool is designed for handling datasets fitting into memory. I fully understand that I'm...

Jan 3 6



Josef Machytka

Quick and Easy Data Exports to Parquet Format Using DuckDB

The Parquet format has become almost an industry standard for Data Lakes and Data Lakehouses, thanks to its efficiency and compact storage...

Dec 5, 2024 17

```
ate the table
 TABLE special_data_types (
 INT AUTO_INCREMENT PRIMARY KEY,
me VARCHAR(50) NOT NULL,
atus ENUM('active', 'inactive', 'pending') NOT NULL,
rmissions SET('read', 'write', 'execute') NOT NULL,
all_number TINYINT NOT NULL,
dium_number MEDIUMINT NOT NULL,
scription TEXT,
ta BLOB,
eated_at DATE NOT NULL


ert 10 rows of data
 INTO special_data_types (name, status, permissions, small_number, medium_number, description, data, created_at)
e', 'active', 'read,write', 5, 1000, 'Alice description', 'Alice data', '2023-01-01'),
, 'inactive', 'read', 10, 2000, 'Bob description', 'Bob data', '2023-02-01'),
lie', 'pending', 'write,execute', 15, 3000, 'Charlie description', 'Charlie data', '2023-03-01'),
d', 'active', 'read,write,execute', 20, 4000, 'David description', 'David data', '2023-04-01'),
, 'inactive', 'execute', 25, 5000, 'Eve description', 'Eve data', '2023-05-01'),
k', 'pending', 'read,write', 30, 6000, 'Frank description', 'Frank data', '2023-06-01'),
e', 'active', 'read', 35, 7000, 'Grace description', 'Grace data', '2023-07-01'),
', 'inactive', 'write,execute', 40, 8000, 'Hank description', 'Hank data', '2023-08-01'),
, 'pending', 'read,write,execute', 45, 9000, 'Ivy description', 'Ivy data', '2023-09-01'),
'. 'active', 'execute', 50, 10000, 'Jack description', 'Jack data', '2023-10-01');
```

 Josef Machytka

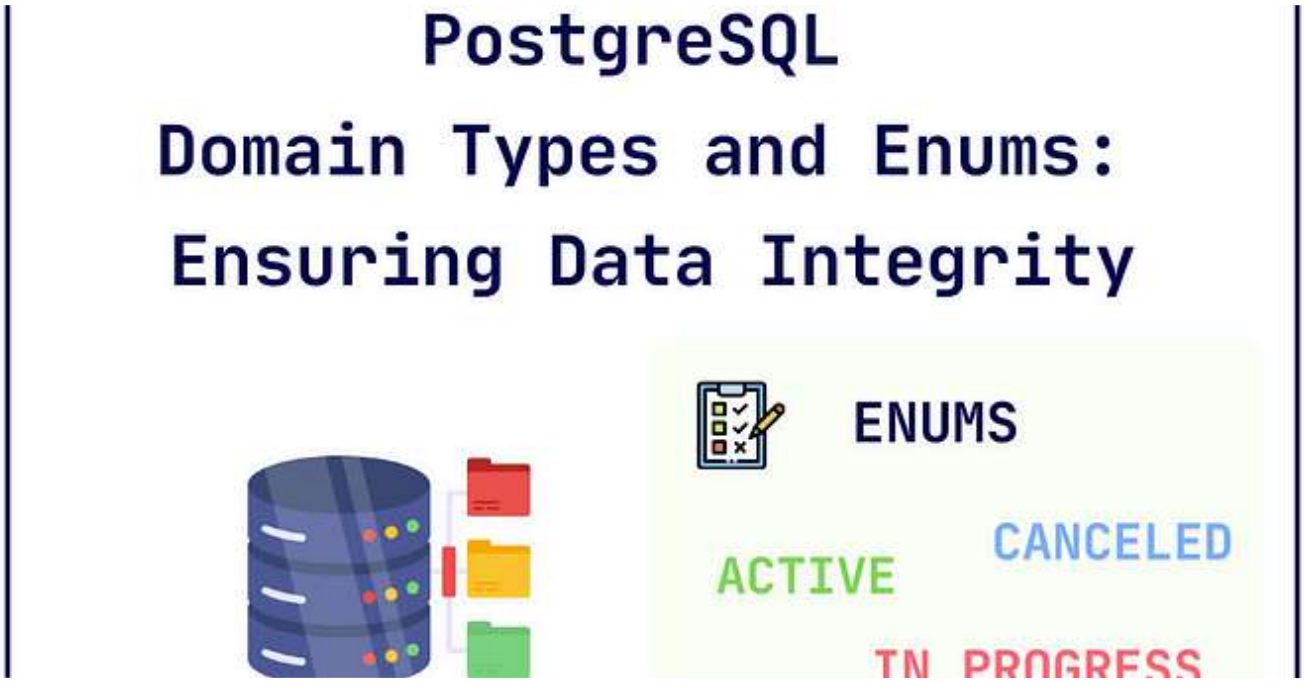
DuckDB as a Rudimentary Data Migration Tool

After exploring how to use DuckDB as an intelligent ETL tool for PostgreSQL, and how to extend its ETL capabilities with simple Python...

Nov 30, 2024 26 2

See all from Josef Machytka

Recommended from Medium



Java Jedi

PostgreSQL Domain Types and Enums: Ensuring Data Integrity

Welcome back to the third post in our series on advanced PostgreSQL features. In the previous posts, we explored how to design a database...

Jan 6 14

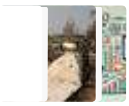
In Towards Data Science by Jiayan Yin

Advanced SQL Techniques for Unstructured Data Handling

Everything you need to know to get started with text mining

Jan 8 126 2

Lists



Staff picks

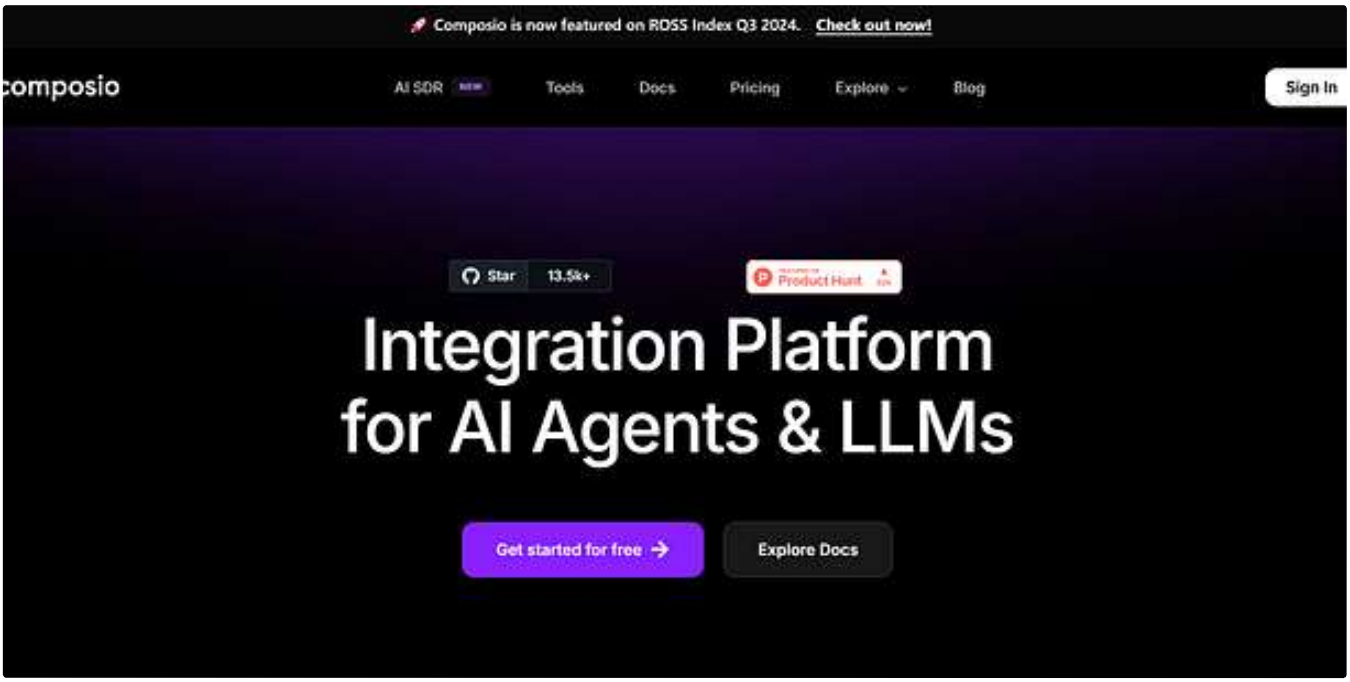
800 stories · 1569 saves

Talaat Magdy

Recursive Queries in PostgreSQL

Recursive queries are one of PostgreSQL's most versatile tools, enabling developers to work with hierarchical or tree-structured data...

Jan 4 28

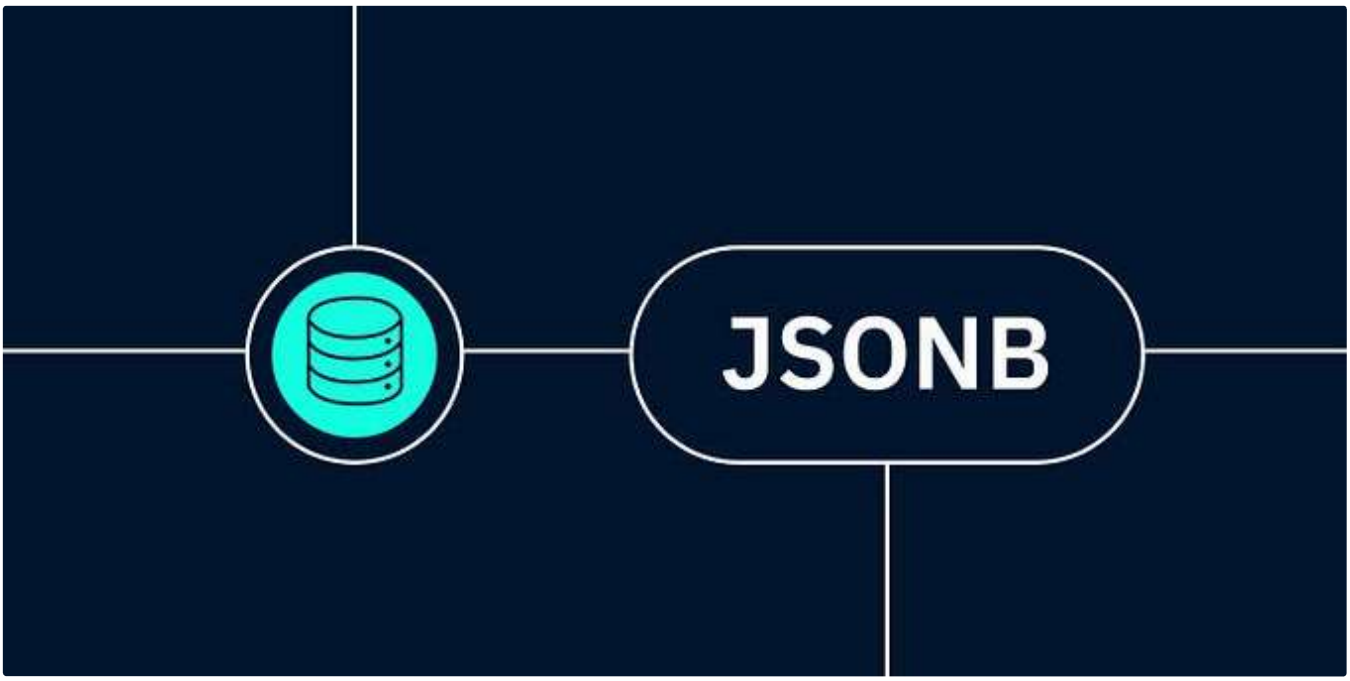


Let's Code Future

15 Modern Open-Source Tools for Supercharge Your Next Project

13 open-source tools to supercharge your next big project this year!

6d ago 246

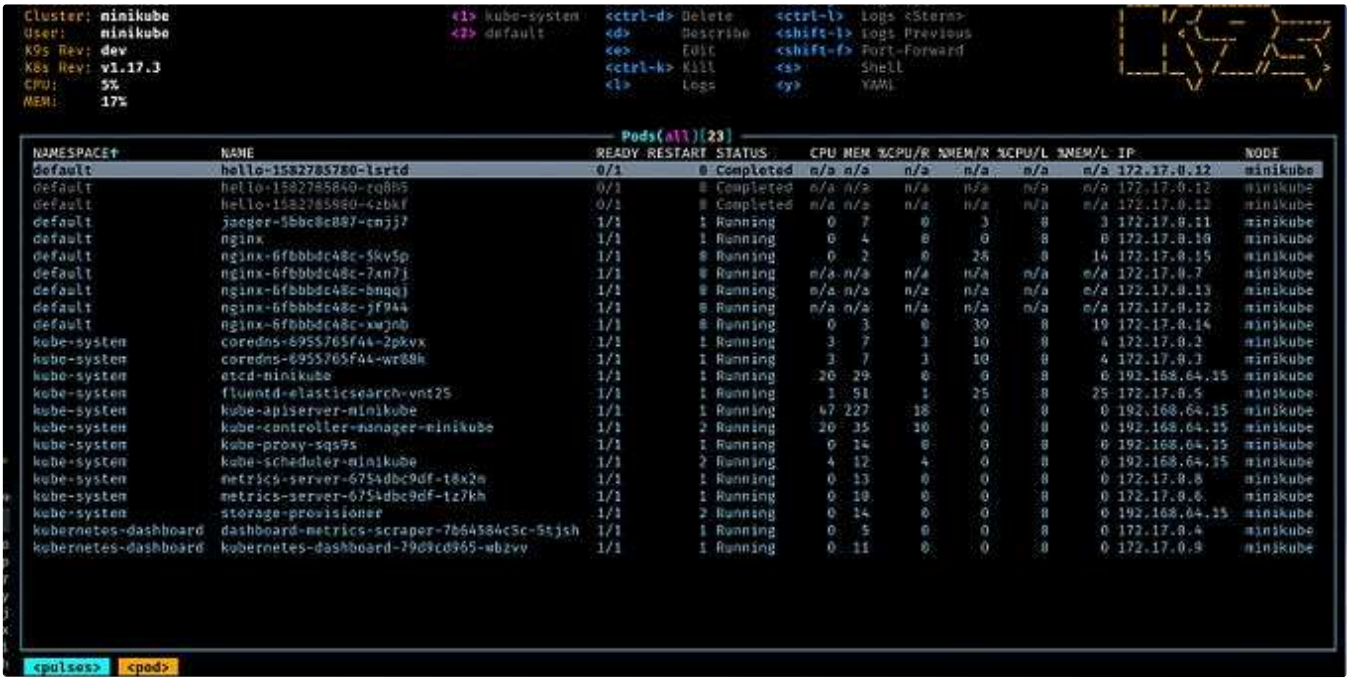


In Mirakl Tech Blog by Mirakl Labs

How JSONB can improve your PostgreSQL queries

By Thomas Fouan Software Engineer at Mirakl

Dec 3, 2024 30



In ITNEXT by Alex Pliutau

Essential CLI/TUI Tools for Developers

An opinionated list of CLI/TUI applications for developer productivity.

Jan 7 572 7

See more recommendations