A PostgreSQL 19 Whirlwind Tour

Jonathan Katz, Sr. Staff Product Manager - Databricks

October 20, 2026

©2026 Databricks, Inc. — All rights reserved

Agenda

- How PostgreSQL is made
- 🌪️ A PostgreSQL 19 Whirlwind Tour
- What's Next

©2026 Databricks, Inc. — All rights reserved

How is PostgreSQL made?

©2026 Databricks, Inc. — All rights reserved

Very Carefully

©2026 Databricks, Inc. — All rights reserved

A whirlwind history of PostgreSQL

- Started in 1980s at UC Berkeley as the POSTGRES project
- 30+ years of open source development! (1996)
  - 30 Years of PostgreSQL Retrospective (PConf.dev 2026)
- Developed by a worldwide community of developers
- No single organization or owner of the project

©2026 Databricks, Inc. — All rights reserved

PostgreSQL Development Cycle

Commitfest = focus on patch review/commits

July	September	November	January	March/April
Commitfest #1	Commitfest #2	Commitfest #3	Commitfest #4	Commitfest #5
			Developer Meeting	

Beta = focus on stabilization

May	June	August	September	Sept / Oct
Beta 1	Beta 2	Beta 3	Release Candidate	GA
Developer Meeting	Release Branch			

14 – 15 Months

©2026 Databricks, Inc. — All rights reserved

Something happened this year

©2026 Databricks, Inc. — All rights reserved

Circa February 2026

Are you using $MODEL to generate real code? I'm...now doing that...

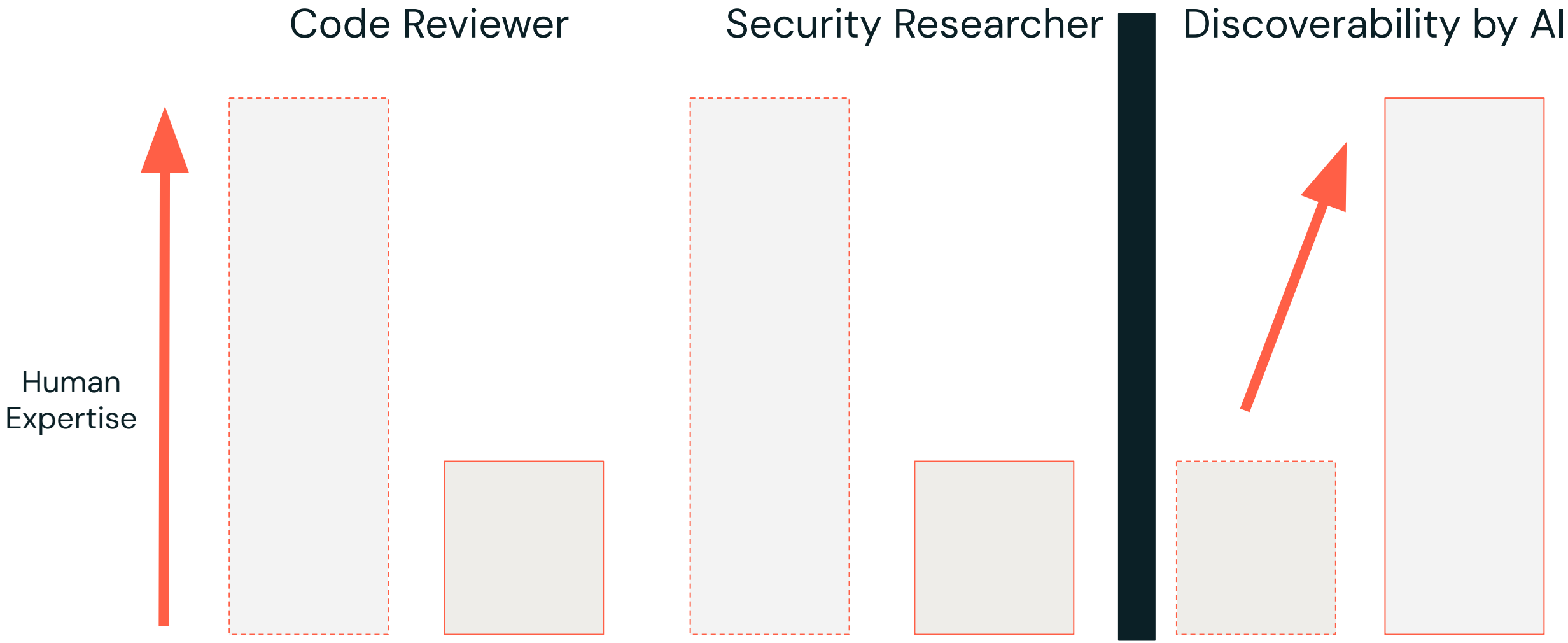
Hey, is it me or is $MODEL like pretty good now?

$MODEL is now actually helping me to move faster.

I'm now using $MODEL to put code in production all the time and it's no longer as buggy.

©2026 Databricks, Inc. — All rights reserved

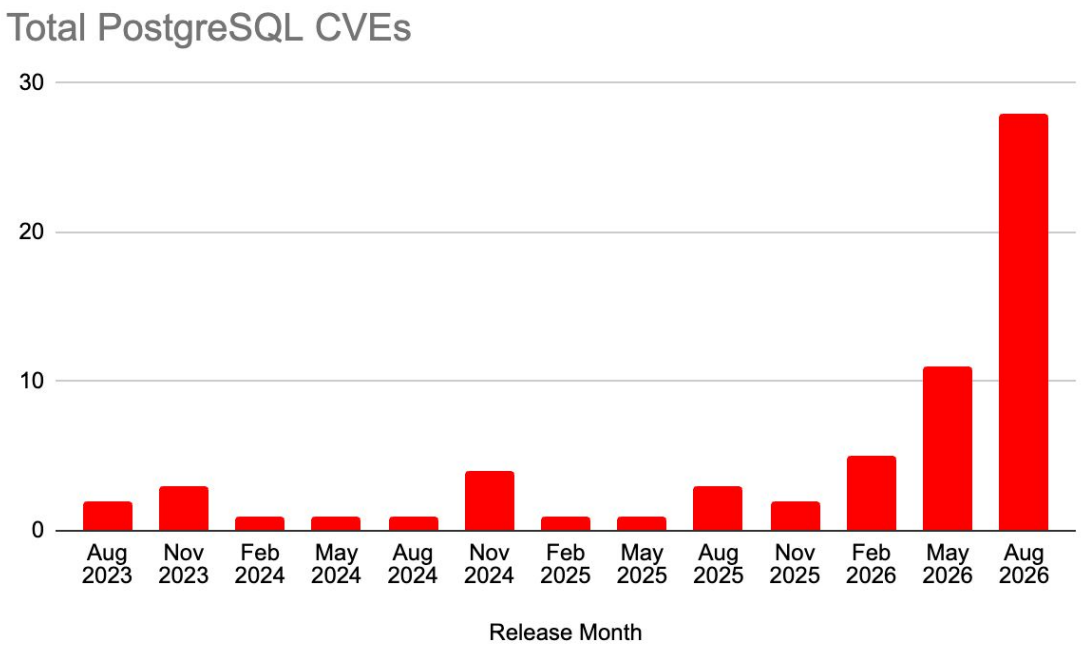
AI Skill acceleration



©2026 Databricks, Inc. — All rights reserved

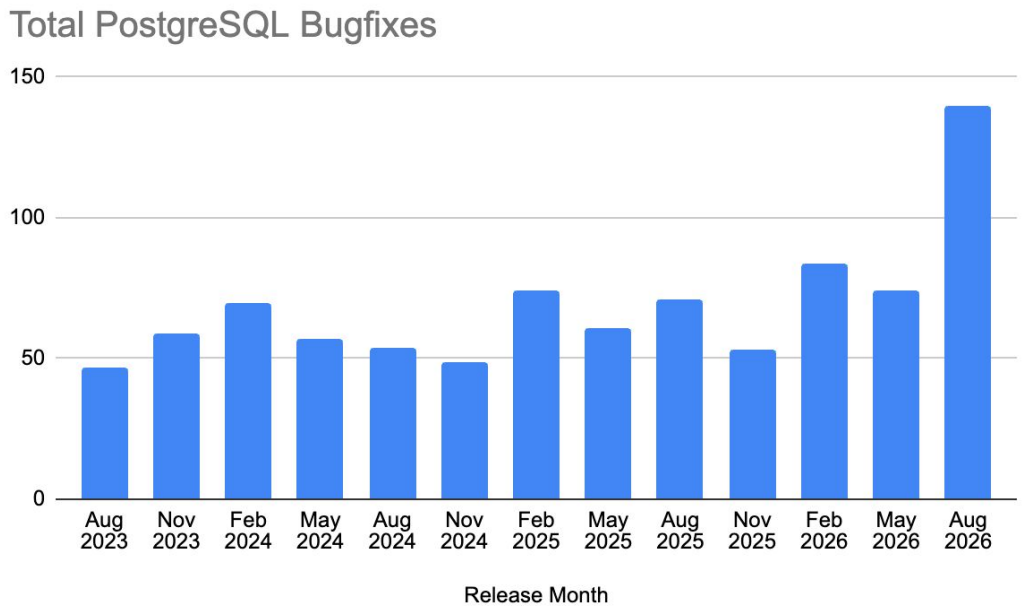
What does this mean for PostgreSQL?

Significant jump in CVE reports



(Varying theoretical/actual severities)

Jump in bug reports

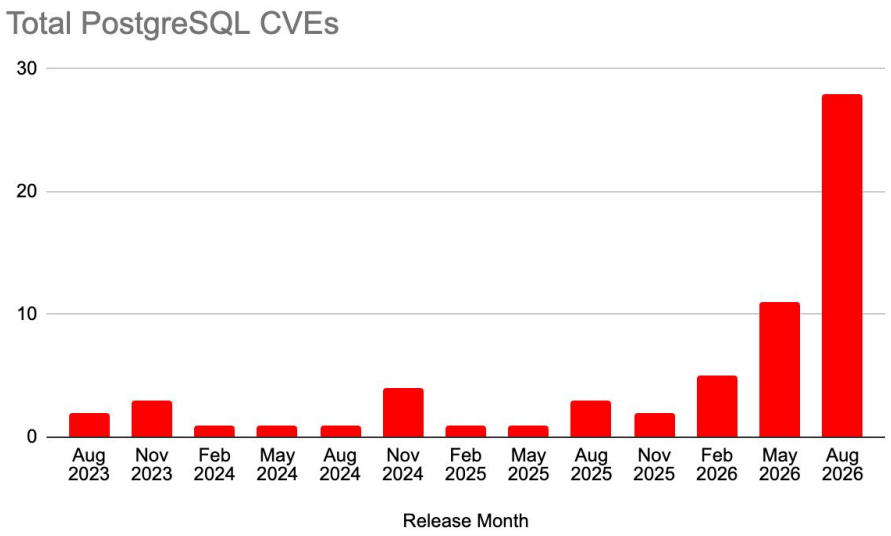


(My opinion: many of these are longtail)

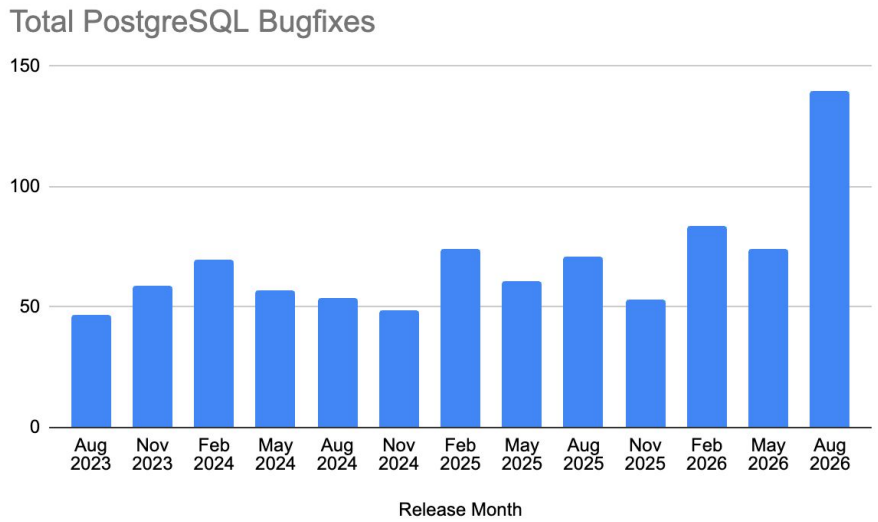
©2026 Databricks, Inc. — All rights reserved

What does this mean for PostgreSQL?

Significant jump in CVE reports



Jump in bug reports



Core PostgreSQL ethos: **reliability and predictability**

- Aim for regular major release cycle date
- Defer / delay some features to a future major version

©2026 Databricks, Inc. — All rights reserved

Some beta features not in PostgreSQL 19

- SQL/PGQ (property graph queries)
  - You can still design and query graph structures in PostgreSQL => extra effort
- Temporal updates/deletes using the `FOR PORTION OF` clause
- `GROUP BY ALL`
- `ALTER TABLE … MERGE/SPLIT PARTITIONS`
- `pg_get_*_ddl()` functions
- Online enablement of data checksums

- Note: some features were omitted prior to the feature freeze

©2026 Databricks, Inc. — All rights reserved

PostgreSQL releases as “layers”

- Partitioning
  - Declarative partitioning introduced in PostgreSQL 10 - every release has added more capabilities and performance improvements
- Logical replication
  - Basis in PostgreSQL 9.4; capability in PostgreSQL 10 - every release has added more capabilities, reliability, and performance improvements
- I/O improvements + building towards Direct I/O
  - PostgreSQL 16 introduced `pg_stat_io` to track I/O usage; 17 streaming I/O, 18 async I/O

©2026 Databricks, Inc. — All rights reserved

PostgreSQL 19
A Whirlwind Tour

©2026 Databricks, Inc. — All rights reserved

PostgreSQL 19 themes

- Improving PostgreSQL workload management
- Enhancements for distributed data
- Improving the developer experience
- Notable changes

©2026 Databricks, Inc. — All rights reserved

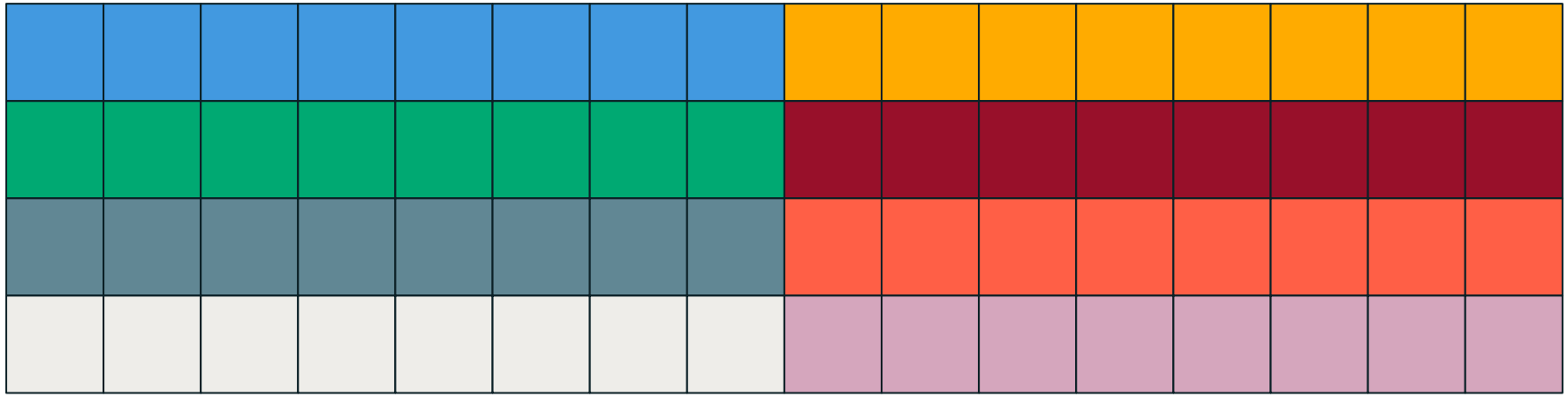
Workload Management

©2026 Databricks, Inc. — All rights reserved

Table Repacking

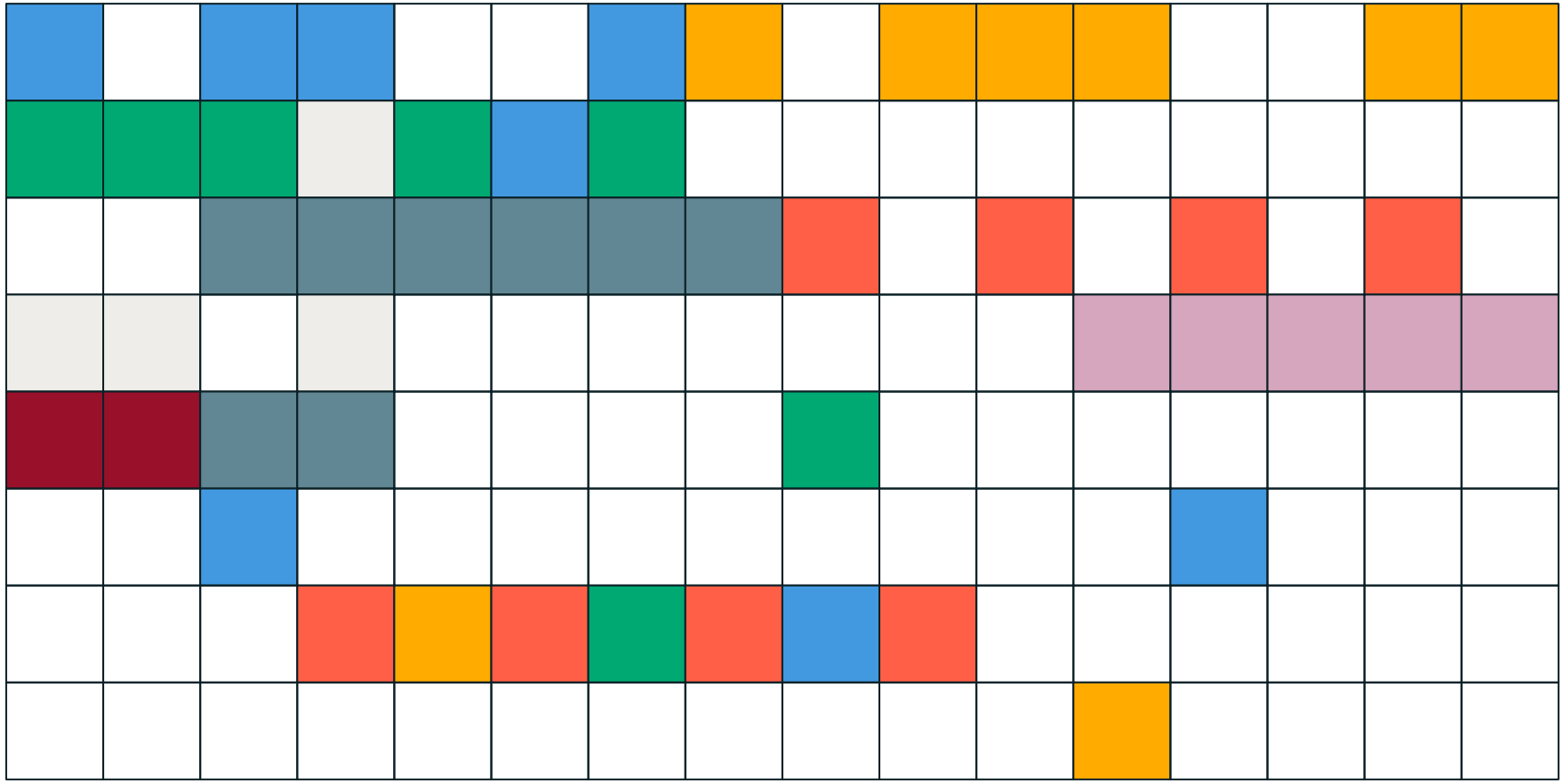
©2026 Databricks, Inc. — All rights reserved

Background – Impact of UPDATE/DELETE



©2026 Databricks, Inc. — All rights reserved

Background – Impact of UPDATE/DELETE



"Bloat"

©2026 Databricks, Inc. — All rights reserved

PostgreSQL table reorganization

- Physical reorganization can help reclaim disk space and improve query performance
- `VACUUM FULL`
  - Removes extra disk space, but requires extra disk space during run
  - Focus on reducing space; no special ordering
  - Fully locks table while running, preventing changes
- `CLUSTER`
  - Reorganizes data around a particular order (e.g. an index)
  - Fully locks table while running, preventing changes
- Several utilities created to help with this challenge (`pg_reorg`, `pg_repack`, `pg_squeeze`)

©2026 Databricks, Inc. — All rights reserved

PostgreSQL 19 lets you REPACK well

- `REPACK` reclaims storage space and can organize data around an ordering without using extra space

- `REPACK CONCURRENTLY` does this **without blocking writes**

```
REPACK (CONCURRENTLY) tbl_name USING INDEX tbl_idx
```

©2026 Databricks, Inc. — All rights reserved

```
SELECT pg_size_pretty(pg_table_size('messages'))   AS table_size,
       pg_size_pretty(pg_indexes_size('messages')) AS index_size;
```

```
 table_size | index_size
------------+------------
 3325 MB    | 1030 MB
```

Example: A chat app's `messages` table: 20M messages, then a cleanup job deleted 75%.

```
SELECT tuple_count, tuple_percent, dead_tuple_count, free_percent
FROM pgstattuple('messages');
```

```
 tuple_count | tuple_percent | dead_tuple_count | free_percent
-------------+---------------+------------------+--------------
     5009438 |         23.33 |             1074 |        73.43
```

©2026 Databricks, Inc. — All rights reserved

Challenge: While messages should be clustered together, they're fragmented across storage

```
EXPLAIN (ANALYZE, COSTS OFF)
SELECT message_id, sender_id, sent_at, body
FROM messages
WHERE conversation_id = 4242
ORDER BY sent_at DESC
LIMIT 50;
```

```
Limit (actual time=0.009..0.112 rows=50.00 loops=1)
  Buffers: shared hit=14 read=41
  ->  Index Scan Backward using
messages_conversation_id_sent_at_idx on messages …
        Index Cond: (conversation_id = 4242)
```

©2026 Databricks, Inc. — All rights reserved

```
REPACK (CONCURRENTLY, ANALYZE, VERBOSE) messages
    USING INDEX messages_conversation_id_sent_at_idx;


INFO:  repacking "public.messages" using sequential scan and sort
INFO:  "public.messages": found 0 removable, 5009997 nonremovable row
versions in 425446 pages
…
INFO:  analyzing "public.messages"
…
REPACK
Time: 4918.634 ms (00:04.919)

EXPLAIN (ANALYZE, COSTS OFF)
SELECT message_id, sender_id, sent_at, body
FROM messages
WHERE conversation_id = 4242
ORDER BY sent_at DESC
LIMIT 50;
```

```
 Limit (actual time=0.046..0.094 rows=50.00 loops=1)
   Buffers: shared hit=7 read=1 dirtied=2
   ->  Index Scan Backward using messages_conversation_id_sent_at_idx on
messages …
```

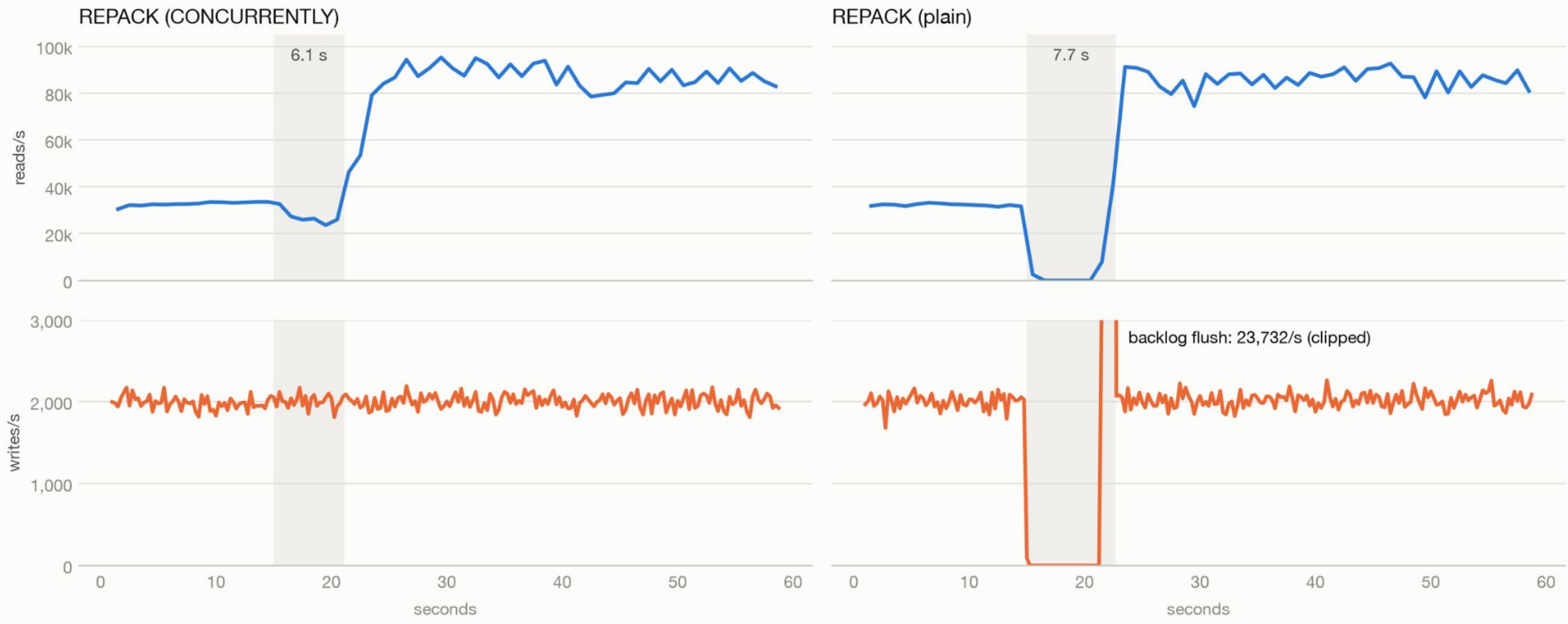
©2026 Databricks, Inc. — All rights reserved

```sql
SELECT message_id, sender_id, sent_at, body
FROM messages
WHERE conversation_id = 4242
ORDER BY sent_at DESC
LIMIT 50;
```

	Before Repack	After Repack
Table size	3,325 MB	833 MB
Index size	1,030 MB	258 MB
Free space	73.4%	1%
Buffers for "last 50 messages"	55	8

©2026 Databricks, Inc. — All rights reserved

Same table, same traffic: CONCURRENTLY keeps serving reads and writes



©2026 Databricks, Inc. — All rights reserved

Things to be aware of…

- `REPACK CONCURRENTLY` has several requirements (must have available replication slot, `PRIMARY KEY` or full replica identity, etc.) - see docs

- Data clustering isn't maintained post-`REPACK`

- `CONCURRENTLY` repacked tables with heavy changes may take a longer lock during the switchover - test accordingly

©2026 Databricks, Inc. — All rights reserved

For a deeper dive, see:

Table Repacking, Done Right – Álvaro Herrera

REPACK in Core – Robert Treat

Tuesday, October 20 at 13:45–14:35

Auditorium 2

©2026 Databricks, Inc. — All rights reserved

Query Plan Management

©2026 Databricks, Inc. — All rights reserved

Background – Anatomy of a Query

SQL is a declarative language

```
SELECT u.username
FROM users u
JOIN users_metadata m ON m.user_id = u.id
WHERE m.geo = 'US'
GROUP BY u.username
ORDER BY m.created_at DESC
LIMIT 25;
```

Imperative

- Go to the users_metadata table
- Loop through all the rows, only choose the ones where geo = "US". Store the user _id in a list with its created_at.
- Go to the users table
- Loop through all the rows, and only choose the ones where the id is in the users_metadata list
- As you iterate, track the username and the created_at value. Keep only the top 25 values.

©2026 Databricks, Inc. — All rights reserved

Background – Anatomy of a Query

SQL is a declarative language

```
SELECT u.username
FROM users u
JOIN users_metadata m ON m.user_id = u.id
WHERE m.geo = 'US'
GROUP BY u.username
ORDER BY m.created_at DESC
LIMIT 25;
```

Declarative

Find the 25 most recent usernames that were created in our US geographic region

A query planner must determine the imperative (execution) plan

©2026 Databricks, Inc. — All rights reserved

How the PostgreSQL query planner works

- ...we don't have enough time for this :-)

- Primarily uses a "cost-based optimizer" (CBO) based on underlying metadata statistics around the data
  - Some rules-based mechanisms for partitioning

- Guiding principle: The PostgreSQL query planner will make a better choice than you.
  - Most of the time

©2026 Databricks, Inc. — All rights reserved

Table Name	Rows
regions	10
warehouses	50
countries	100
categories	500
suppliers	10,000
products	200,000
product_reviews	1,000,000
customers	2,000,000
inventory	4,000,000
orders (14 daily partitions)	10,000,000

- **Scenario**: 2,000 enterprise customers (0.1%) place 30% of all orders.

- No per-table statistic can express that, because it only shows up across the `customers` / `orders` join
  - (`CREATE STATISTICS` only covers columns of one table.)

©2026 Databricks, Inc. — All rights reserved

“Find the 5 warehouses with the most at risk orders for our enterprise customers”

```sql
SELECT w.name AS warehouse, count(*) AS at_risk_orders
FROM customers c
JOIN orders o     ON o.customer_id = c.customer_id
JOIN inventory i  ON i.warehouse_id = o.warehouse_id
                 AND i.product_id = o.product_id
JOIN warehouses w ON w.warehouse_id = o.warehouse_id
WHERE c.segment = 'enterprise'
  AND o.order_date >= '2026-09-08'
  AND i.quantity < 100
GROUP BY w.name
ORDER BY at_risk_orders DESC
LIMIT 5;
```

```
  warehouse    | at_risk_orders
---------------+----------------
 Warehouse 34 |           1296
 Warehouse 12 |           1291
 Warehouse 41 |           1285
 Warehouse 29 |           1282
 Warehouse 33 |           1268
(5 rows)

Time: 921.849 ms
```

©2026 Databricks, Inc. — All rights reserved

```sql
EXPLAIN (ANALYZE)
SELECT w.name AS warehouse, count(*) AS at_risk_orders
FROM customers c
JOIN orders o     ON o.customer_id = c.customer_id
JOIN inventory i  ON i.warehouse_id = o.warehouse_id
                 AND i.product_id = o.product_id
JOIN warehouses w ON w.warehouse_id = o.warehouse_id
WHERE c.segment = 'enterprise'
  AND o.order_date >= '2026-09-08'
  AND i.quantity < 100
GROUP BY w.name
ORDER BY at_risk_orders DESC
LIMIT 5;
```

```
->  Nested Loop  (… rows=105 …) (actual … rows=19969.67 loops=3)
      ->  Hash Join  (… rows=2708 …) (actual … rows=501204.00 loops=3)
            Hash Cond: (o.customer_id = c.customer_id)
            …
      ->  Index Scan using inventory_pkey on inventory i  (… rows=1 …)
                                                (actual … rows=0.04 loops=1503612)
            Index Cond: ((warehouse_id = o.warehouse_id) AND
                         (product_id = o.product_id))
            Filter: (quantity < 100)
…
Execution Time: 1017.892 ms
```

©2026 Databricks, Inc. — All rights reserved

How do you deal with a bad plan?

- …we also don’t have enough time for this :-)
- Methods include:
  - Rewriting queries
  - `ANALYZE`
  - Modifying costs
  - Disabling scans
  - Modifying schema
  - Modifying other configuration
  - Asking for help
  - Migrating away (and migrating back)
  - ~~Hints~~

©2026 Databricks, Inc. — All rights reserved

PostgreSQL 19: Query plan management

- Introduces two extensions for user management of query plans
  - These aren't hints - they're better (IMV)

- `pg_plan_advice`: Provides "advice language" for inspecting and working to stabilize plans

- `pg_stash_advice`: Stores and automatically applies advice

- Both need to be present in `shared_preload_libraries`
  - `pg_plan_advice` automatically available
  - `CREATE EXTENSION pg_stash_advice;`

©2026 Databricks, Inc. — All rights reserved

```
EXPLAIN (COSTS OFF, PLAN_ADVICE)
SELECT w.name AS warehouse, count(*) AS at_risk_orders
FROM customers c
JOIN orders o     ON o.customer_id = c.customer_id
JOIN inventory i  ON i.warehouse_id = o.warehouse_id
                 AND i.product_id = o.product_id
JOIN warehouses w ON w.warehouse_id = o.warehouse_id
WHERE c.segment = 'enterprise'
  AND o.order_date >= '2026-09-08'
  AND i.quantity < 100
GROUP BY w.name
ORDER BY at_risk_orders DESC
LIMIT 5;
```

```
Generated Plan Advice:
   JOIN_ORDER(o c i w)
   NESTED_LOOP_PLAIN(i)
   NESTED_LOOP_MATERIALIZE(w)
   HASH_JOIN(c)
   SEQ_SCAN(o/public.orders_2026_09_08 o/public.orders_2026_09_09
    o/public.orders_2026_09_10 o/public.orders_2026_09_11
    o/public.orders_2026_09_12 o/public.orders_2026_09_13
    o/public.orders_2026_09_14 w)
   INDEX_SCAN(c public.customers_segment_idx i public.inventory_pkey)
   PARTITIONWISE(o)
   GATHER_MERGE((c o i w))
```

©2026 Databricks, Inc. — All rights reserved

```
SET pg_plan_advice.advice = 'HASH_JOIN(i)';

EXPLAIN (ANALYZE, COSTS OFF)
SELECT w.name AS warehouse, count(*) AS at_risk_orders
FROM customers c
JOIN orders o     ON o.customer_id = c.customer_id
JOIN inventory i  ON i.warehouse_id = o.warehouse_id
                 AND i.product_id = o.product_id
JOIN warehouses w ON w.warehouse_id = o.warehouse_id
WHERE c.segment = 'enterprise'
  AND o.order_date >= '2026-09-08'
  AND i.quantity < 100
GROUP BY w.name
ORDER BY at_risk_orders DESC
LIMIT 5;
```

```
->  Parallel Hash Join (actual … rows=19969.67 loops=3)
      Hash Cond: ((o.warehouse_id = i.warehouse_id) AND (o.product_id =
i.product_id))
      ->  Hash Join (actual … rows=501204.00 loops=3)
            …
      ->  Parallel Hash (actual … rows=133390.33 loops=3)
            ->  Parallel Seq Scan on inventory i (actual … rows=133390.33 loops=3)
                  Filter: (quantity < 100)
…
 Supplied Plan Advice:
   HASH_JOIN(i) /* matched */
 Execution Time: 339.347 ms
```

©2026 Databricks, Inc. — All rights reserved

```
CREATE EXTENSION pg_stash_advice;

EXPLAIN (VERBOSE, COSTS OFF) SELECT …;
--    Query Identifier: 5714514445752058280

SELECT pg_create_advice_stash('dashboards');
SELECT pg_set_stashed_advice('dashboards', 5714514445752058280,
'HASH_JOIN(i)');

ALTER DATABASE test SET pg_stash_advice.stash_name = 'dashboards';

SELECT * FROM pg_get_advice_stash_contents('dashboards');
```

```
 stash_name |      query_id       | advice_string
------------+---------------------+---------------
 dashboards | 5714514445752058280 | HASH_JOIN(i)
(1 row)
```

©2026 Databricks, Inc. — All rights reserved

Things to be aware of…

- Prepared statements / driver cached plans may use an old plan after advice changes
  - (May take 5+ runs for this to change)

- Helpful for fixing “plan flips”, but ensure you understand the root cause

- In normal operations, query plans can change as underlying data changes
  - The PostgreSQL query planner is still more often than not correct :)

©2026 Databricks, Inc. — All rights reserved

For a deeper dive, see:
pg_plan_advice: Plan Stability and User Planner Control for PostgreSQL?
– Robert Haas

Thursday, October 22 at 13:55–14:45
Auditorium 1

©2026 Databricks, Inc. — All rights reserved

Other Workload Optimizations

©2026 Databricks, Inc. — All rights reserved

Vacuuming enhancements

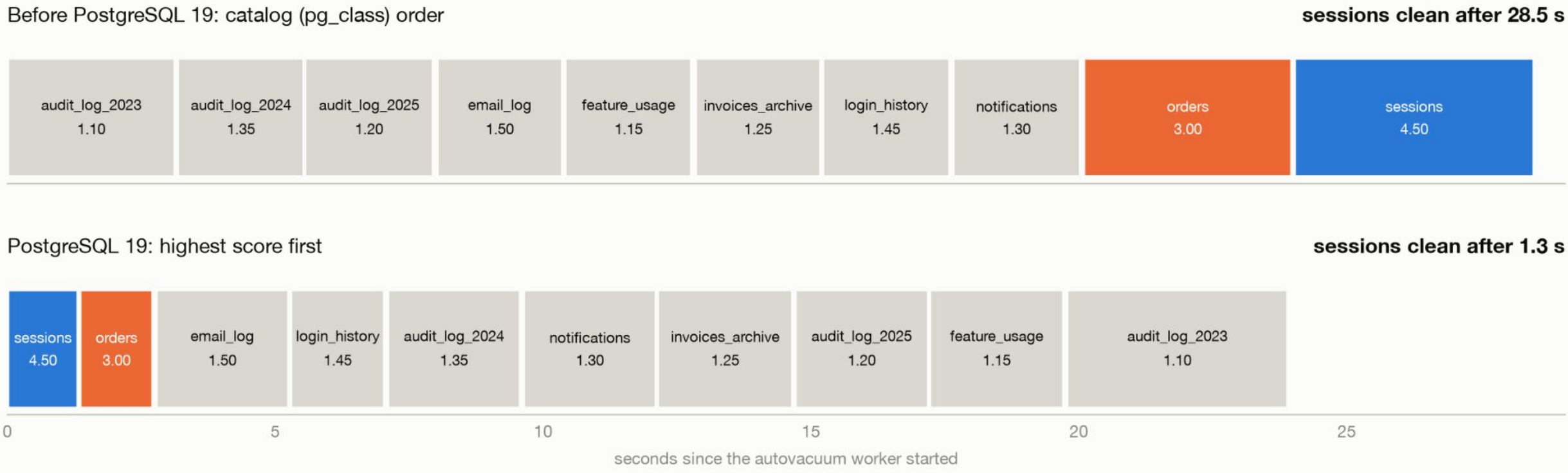
(...so you don't have to REPACK? :-)

- PostgreSQL uses table reads to proactively optimize for later vacuums
- autovacuum can now use parallelism for vacuuming indexes
  - `autovacuum_max_parallel_workers`
- `vacuum --analyze-only` and `vacuum–analyze-in-stages` now analyzes partition tables when no targets are specified
- New autovacuum prioritization system

©2026 Databricks, Inc. — All rights reserved

One autovacuum worker, ten tables that need a vacuum

Same tables, same dead rows. Each block is one table's vacuum, labeled with its score (dead rows ÷ vacuum threshold).



©2026 Databricks, Inc. — All rights reserved

Observability features

- New stats views
  - `pg_stat_lock` (per-lock-type statistics)
  - `pg_stat_recovery`
  - `pg_stat_autovacuum_scores`

- Query-level: `EXPLAIN (ANALYZE, IO)` and `auto_explain.log_io` report asynchronous I/O activity

- Write-ahead log (WAL)
  - Bytes written for full-page images are reported in `pg_stat_wal`, `EXPLAIN (ANALYZE, WAL)` and, vacuum/analyze logs

©2026 Databricks, Inc. — All rights reserved

Enhancements for Distributed Data

©2026 Databricks, Inc. — All rights reserved

Tell me about distributed PostgreSQL

- Synchronization via replication
  - Physical
  - Logical – further customized through decoding plugins

- Federation via foreign data wrappers (FDWs)
  - `postgres_fdw`
  - Many, many others for all sorts of systems: https://wiki.postgresql.org/wiki/Foreign_data_wrappers

©2026 Databricks, Inc. — All rights reserved

Logical Replication – Overview

- Stream changes from PostgreSQL into downstreams systems
  - Downstream systems subscribe and interpret changes

- Use cases:
  - Change-data capture (CDC)
  - Real-time data serving
  - ETL/ELT pipelines
  - Edge systems

- Logical replication has emerged as a critical feature for production systems

©2026 Databricks, Inc. — All rights reserved

PostgreSQL 19 – Logical replication features

- Logical replication can be enabled without restarting the server
  - New setting `effective_wal_level` shows enablement

- Sequences can now be logically replicated
  - `CREATE PUBLICATION migrate FOR ALL TABLES, ALL SEQUENCES;`

- More information for write conflicts via `retain_dead_tuples` setting on `CREATE SUBSCRIPTION`
  - `pg_stat_subscription_stats` counts `update_deleted` conflicts

- `CREATE PUBLICATION … EXCLUDE` lets you publish all tables EXCEPT the ones listed

©2026 Databricks, Inc. — All rights reserved

Using SERVER for logical replication setup

```
CREATE SERVER publisher
    FOREIGN DATA WRAPPER postgres_fdw
    OPTIONS (port '5433', dbname 'test2');

CREATE USER MAPPING FOR CURRENT_USER
    SERVER publisher
    OPTIONS (user 'jkatz', password 's3cret?');

CREATE SUBSCRIPTION orders_sub
    SERVER publisher
    PUBLICATION orders_pub;
```

©2026 Databricks, Inc. — All rights reserved

Read-your-writes consistency

- **Challenge**: Offload reads to a physical read replica, but can't read the data just committed without challenging app code.

```
-- PRIMARY: user 42 adds an item to their cart
INSERT INTO cart_items (user_id, product_id)
VALUES (42, 1001)
RETURNING cart_item_id;              -- 1499625
```

```
-- STANDBY: the app reads the cart from a replica
SELECT cart_item_id, user_id, product_id
FROM cart_items
WHERE cart_item_id = 1499625;

 cart_item_id | user_id | product_id
--------------+---------+------------
(0 rows)
```

©2026 Databricks, Inc. — All rights reserved

Read-your-writes consistency

- `WAIT FOR LSN` – instruct connection to standby to not return until data is available on replica

```
-- PRIMARY: user 42 adds an item to their cart
INSERT INTO cart_items (user_id, product_id)
VALUES (42, 1001)
RETURNING cart_item_id;              -- 1499625

SELECT pg_current_wal_insert_lsn(); -- 1/87299680
```

```
-- STANDBY: the app reads the cart from a replica
WAIT FOR LSN '1/87299680';

SELECT cart_item_id, user_id, product_id
FROM cart_items
WHERE cart_item_id = 1499625;

 cart_item_id | user_id | product_id
--------------+---------+------------
      1499625 |      42 |       1001
(1 row)
```

©2026 Databricks, Inc. — All rights reserved

For a deeper dive, see:
WAIT FOR the Feature for Almost 10 Years
– Alexander Korotkov

Tuesday, October 20 at 13:05–13:35
Auditorium 2

©2026 Databricks, Inc. — All rights reserved

FDWs: Table statistics from remote systems

- **Recall**: PostgreSQL uses a cost-based optimizer for query planning

- Before v19, PostgreSQL couldn't import the remote statistics
  - Could do remote sampling

```
CREATE SERVER loyalty
  FOREIGN DATA WRAPPER postgres_fdw
    OPTIONS (port '5433', dbname 'test2');

CREATE USER MAPPING
  FOR CURRENT_USER
  SERVER loyalty;

IMPORT FOREIGN SCHEMA public
  LIMIT TO (loyalty_accounts)
    FROM SERVER loyalty INTO public;

-- show stats on remote table
SELECT relname, reltuples
FROM pg_class WHERE relname = 'loyalty_accounts';
```

```
     relname      | reltuples
------------------+-----------
 loyalty_accounts |        -1
```

©2026 Databricks, Inc. — All rights reserved

FDWs: Table statistics from remote systems

```
EXPLAIN (ANALYZE)
SELECT
    la.tier, count(*) AS orders,
    sum(o.amount) AS revenue
FROM orders o
JOIN loyalty_accounts la ON
    la.customer_id = o.customer_id
WHERE o.order_date >= '2026-09-12'
GROUP BY la.tier
ORDER BY revenue DESC;
```

```
->  Nested Loop  (… rows=36937 …) (actual … rows=2142854.00 loops=1)
      ->  Foreign Scan on loyalty_accounts la  (… rows=1280 …)
                                        (actual … rows=2000000.00 loops=1)
      ->  Append  (… rows=32 …) (actual … rows=1.07 loops=2000000)
            ->  Index Scan using orders_2026_09_12_customer_id_idx … (actual …
loops=2000000)
            ->  Index Scan using orders_2026_09_13_customer_id_idx … (actual …
loops=2000000)
            ->  Index Scan using orders_2026_09_14_customer_id_idx … (actual …
loops=2000000)
…
Execution Time: 4977.259 ms
```

Without stats, Postgres assumes 10 pages. The table actually has 2,000,000 rows

©2026 Databricks, Inc. — All rights reserved

FDWs: Table statistics from remote systems

```
ALTER SERVER loyalty
    OPTIONS (ADD import_stats 'true');
ANALYZE VERBOSE loyalty_accounts;

EXPLAIN (ANALYZE)
SELECT
    la.tier, count(*) AS orders,
    sum(o.amount) AS revenue
FROM orders o
JOIN loyalty_accounts la ON
    la.customer_id = o.customer_id
WHERE o.order_date >= '2026-09-12'
GROUP BY la.tier
ORDER BY revenue DESC;
```

```
->  HashAggregate  (… rows=4 …) (actual … rows=4.00 loops=1)
      ->  Hash Join  (… rows=2142854 …) (actual … rows=2142854.00 loops=1)
            Hash Cond: (o.customer_id = la.customer_id)
            ->  Append  (… rows=2142854 …) (actual … rows=2142854.00 loops=
                  ->  Seq Scan on orders_2026_09_12 o_1 …
                  ->  Seq Scan on orders_2026_09_13 o_2 …
                  ->  Seq Scan on orders_2026_09_14 o_3 …
            ->  Hash  (… rows=1999993 …) (actual … rows=2000000.00 loops=1)
                  ->  Foreign Scan on loyalty_accounts la  (… rows=1999993
                                                (actual … rows=2000000.00 loops=
…
Execution Time: 1692.315 ms
```

©2026 Databricks, Inc. — All rights reserved

Improving the developer experience

©2026 Databricks, Inc. — All rights reserved

Conveniences for developers

- Additional string functions for the `jsonpath` SQL/JSON path language

```
SELECT jsonb_path_query('{"email": "
FirstName.LastName@Example.COM "}',
'$.email.btrim().lower().split_part("@", 2)');
```

```
 jsonb_path_query
------------------
 "example.com"
```

- `random` supports dates and timestamps

```
SELECT
    random(DATE '2026-01-01', DATE '2026-12-31') AS d,
    random(TIMESTAMP '2026-01-01', TIMESTAMP '2026-12-31') AS ts,
    random(now() - interval '1 day', now()) AS tstz;
```

©2026 Databricks, Inc. — All rights reserved

Conveniences for developers

- "UPSERT" (`INSERT .. ON CONFLICT`) now supports the `get_or_create` pattern

```
CREATE TABLE users (
    id int GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    email text UNIQUE
);
INSERT INTO users (email) VALUES ('jkatz@example.com');

INSERT INTO users (email) VALUES ('jkatz@example.com'),
('another@example.com')
ON CONFLICT (email) DO SELECT
RETURNING id, email, old.id IS NULL AS created;

 id |        email        | created
----+---------------------+---------
  1 | jkatz@example.com   | f
  5 | another@example.com | t
```

©2026 Databricks, Inc. — All rights reserved

Conveniences for developers

- COPY can write out to JSON directly

```
CREATE TABLE users (
    id int GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    email text UNIQUE,
    tags text[],
    last_login timestamptz);
INSERT INTO users (email, tags, last_login)
VALUES
    ('jkatz@example.com', '{admin,dba}', '2026-09-29 10:15:00-04'),
    ('new@example.com', '{}', NULL);
```

```
COPY users TO STDOUT (FORMAT JSON, FORCE_ARRAY);
```

```
[
    {"id":1,"email":"jkatz@example.com","tags":["admin","dba"],"last_login":"2026-09-29T10:15:00-04:00"},
    {"id":2,"email":"new@example.com","tags":[],"last_login":null}
]
```

©2026 Databricks, Inc. — All rights reserved

Conveniences for developers

- Performance
  - I/O workers automatically scale between `io_min_workers` and `io_max_workers`
  - Quicker `FOREIGN KEY` checks (up to 2x faster)
  - More optimizations to the planner / executor

- `password_expiration_warning_threshold` server parameter warns users when password is expiring

- Event triggers can be written in PL/Python

©2026 Databricks, Inc. — All rights reserved

Notable Changes

©2026 Databricks, Inc. — All rights reserved

Notable Changes

- JIT (just-in-time compilation) disabled by default
- TOAST now uses `lz4` compression by default instead of `pglz`
- md5 password deprecation warnings get louder
- `pg_upgrade` won't upgrade if
  - Using `btree_gist` extension for indexes on `inet` and `cidr` columns
  - Carriage returns (`\r`) in database, role, or tablespace names

©2026 Databricks, Inc. — All rights reserved

Am I Missing Something?

©2026 Databricks, Inc. — All rights reserved

Definitely Yes

©2026 Databricks, Inc. — All rights reserved

https://www.postgresql.org/docs/19/release-19.html

©2026 Databricks, Inc. — All rights reserved

What’s Next?

©2026 Databricks, Inc. — All rights reserved

What's Next?

- November 12, 2026 - PostgreSQL 19 RC2 (along with other update releases)
- November 19, 2026 - PostgreSQL 19 GA (Planned)
- November 20, 2026 - You run it in production ;-)
  - ^ this is also a Friday, a great day to go to production ;-)
- November 13, 2031 - PostgreSQL 19 EOL (we hardly knew you)

©2026 Databricks, Inc. — All rights reserved

Oh you wanted a roadmap?

©2026 Databricks, Inc. — All rights reserved

Resources for ongoing development

- pgsql-hackers@lists.postgresql.org

- https://commitfest.postgresql.org

- https://planet.postgresql.org

©2026 Databricks, Inc. — All rights reserved

Thank You!

©2026 Databricks, Inc. — All rights reserved

©2026 Databricks, Inc. — All rights reserved