Microsoft Azure



Postgres 19, 20, & Beyond:
Live Demos of New Features & Tools



Franck Pachot
Developer Advocate
Principal Product Manager

PG 18.6

Feature	Authorship
https://github.com/pglayers/pglayers	Ismaël Mejía

```
# podman run -d --replace --name PG18 --hostname PG18 \
             -e POSTGRES_PASSWORD=x -e POSTGRES_INITDB_ARGS="--no-data-checksums" \
              ghcr.io/pglayers/pglayers-full:18

# podman exec -it -e PGUSER=postgres PG18 psql -P pager=off

psql (18.6 (Debian 18.6-1.pgdg13+2))

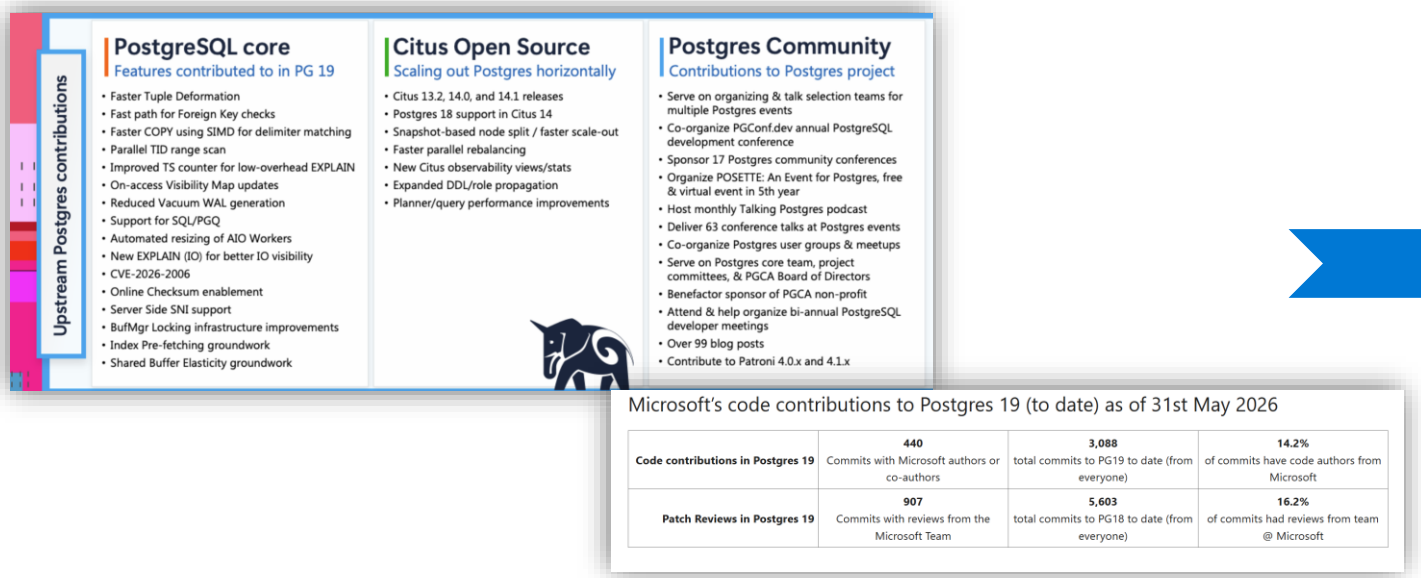
postgres=# select count(*), string_agg(name,', ') from pg_available_extensions()
;
 count | string_agg
-------+-----------------------------------------------------------------------------------------------
   144 | age, anon, credcheck, db2fce, documentdb_extended_rum, documentdb_core, documentdb, pg_documentdb_gw_host, extra_window_functions,
first_last_agg, h3, h3_postgis, hll, http, hypopg, icu_ext, ip4r, jsquery, orafce, periods, pg_background, pg_bigm, pg_clickhouse, pg_cron,
pg_csv, pg_dirtyread, pg_duckdb, pg_durable, pg_graphql, pg_hashids, pg_hint_plan, pg_ivm, pg_jsonschema, pg_net, pg_partman, pg_permissions,
pg_pwhash, pg_qualstats, re2, pg_repack, roaringbitmap, pg_rrule, pg_show_plans, pg_similarity, pg_squeeze, pg_stat_monitor, pg_statviz,
pg_textsearch, pg_uuidv7, pg_wait_sampling, pgaudit, pgfincore, pgjwt, pglogical, pglogical_origin, pgmq, pgnodemx, pg_proctab, pgpcre,
pgrouting, pgsodium, pg_sphere, pgtap, pgtt, vector, vectorscale, pljs, plpgsql_check, plprofiler, plv8, postgis, postgis_raster-3,
postgis_topology, address_standardizer, address_standardizer_data_us, address_standardizer-3, address_standardizer_data_us-3, postgis_sfcgal,
postgis_raster, postgis_tiger_geocoder, postgis_topology-3, postgis_tiger_geocoder-3, postgis-3, postgis_sfcgal-3, postgres_protobuf, prefix,
prioritize, pg_rational, rum, semver, set_user, tdigest, tds_fdw, temporal_tables, timescaledb, timestamp9, toastinfo, wrappers, lo, uuid-
ossp, pgrowlocks, amcheck, pg_prewarm, seg, tsm_system_rows, btree_gist, bloom, pg_logicalinspect, pg_surgery, pgcrypto, pg_stat_statements,
plpgsql, intagg, dict_xsyn, isn, pg_buffercache, tablefunc, cube, sslinfo, earthdistance, pg_visibility, ltree, insert_username, pg_trgm,
refint, tcn, autoinc, postgres_fdw, dict_int, pg_freespacemap, intarray, file_fdw, hstore, moddatetime, pg_walinspect, dblink, unaccent,
btree_gin, tsm_system_time, xml2, pgstattuple, citext, pageinspect, fuzzystrmatch

postgres=# show extension_control_path;
```



db<>fiddle

Who contributed to PostgreSQL this year (PG19 & 20)?

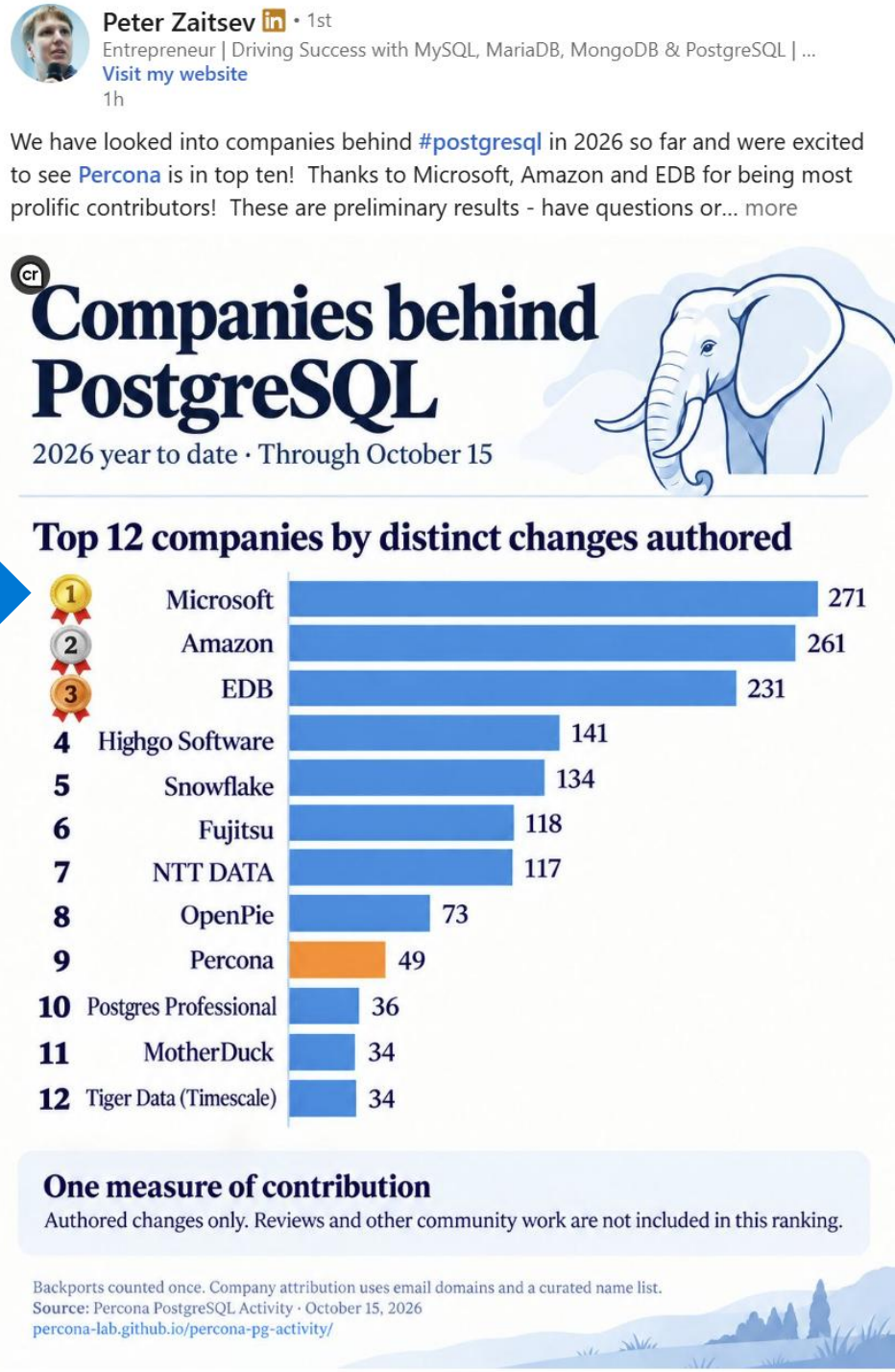


Microsoft's code contributions to Postgres 19 (to date) as of 31st May 2026

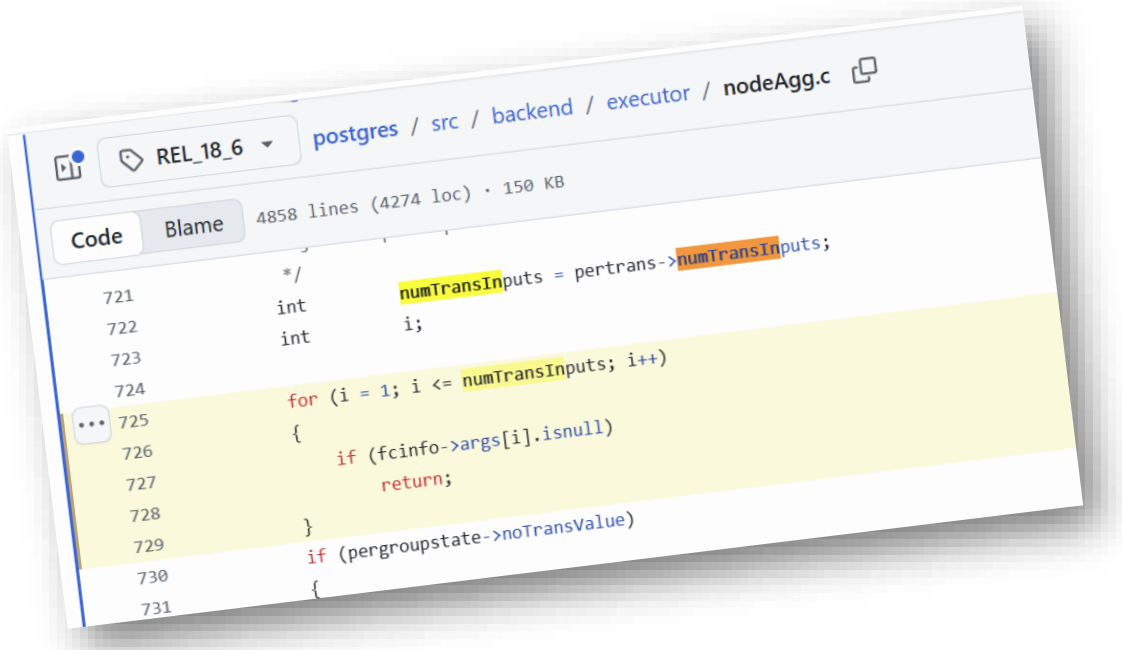
Code contributions in Postgres 19	**440** Commits with Microsoft authors or co-authors	**3,088** total commits to PG19 to date (from everyone)	**14.2%** of commits have code authors from Microsoft
Patch Reviews in Postgres 19	**907** Commits with reviews from the Microsoft Team	**5,603** total commits to PG18 to date (from everyone)	**16.2%** of commits had reviews from team @ Microsoft

I looked at commit fests for:

- Authored patches (Microsoft-funded projects, not review-only)
- Co-authored patches (not committer-only work)
- Many were driven by large enterprise customer needs



What's Faster? COUNT(*) or COUNT(1)?



MySQL: No relevant difference

Oracle: No relevant difference

SQL Server: No relevant difference

PostgreSQL: Significant difference



PG 18.6

```
postgres=# create table demo (
 key   bigint primary key,
 value text
);
CREATE TABLE

postgres=# insert into demo select generate_series(1,1e6),
repeat(md5(random()::text),10)
; vacuum analyze demo
;
INSERT 0 1000000
VACUUM

postgres=# explain (verbose, analyze, buffers, costs off)
 select count(1) from demo
;
                                                    QUERY PLAN
------------------------------------------------------------------------------------------------------------------
Aggregate (actual time=1804.676..1804.680 rows=1.00 loops=1)
   Output: count(1)
   Buffers: shared hit=14 read=2723
   ->  Index Only Scan using demo_pkey on public.demo
       (actual time=0.156..939.177 rows=1000000.00 loops=1)
         Output: key
         Heap Fetches: 0
         Index Searches: 1
         Buffers: shared hit=14 read=2723
```



PG 19 RC1

```
postgres=# create table demo (
 key   bigint primary key,
 value text
);
CREATE TABLE

postgres=# insert into demo select generate_series(1,1e6),
repeat(md5(random()::text),10)
; vacuum analyze demo
;
INSERT 0 1000000
VACUUM

postgres=# explain (verbose, analyze, buffers, costs off)
 select count(1) from demo
;
                                                    QUERY PLAN
------------------------------------------------------------------------------------------------------------------
Aggregate (actual time=1636.153..1636.156 rows=1.00 loops=1)
   Output: count(*)
   Buffers: shared hit=2737
   ->  Index Only Scan using demo_pkey on public.demo
       (actual time=0.043..840.637 rows=1000000.00 loops=1)
         Output: key
         Heap Fetches: 0
         Index Searches: 1
         Buffers: shared hit=2737
```



PG 18.6

```
postgres=# alter table demo alter column value set not null
;
ALTER TABLE

postgres=# explain (verbose, analyze, buffers, costs off)
 select count(value) from demo
;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------
Aggregate (actual time=1859.533..1859.537 rows=1.00 loops=1)
   Output: count(value)
   Buffers: shared hit=13227 read=32228
   ->  Seq Scan on public.demo
       (actual time=0.702..985.037 rows=1000000.00 loops=1)
         Output: key, value
         Buffers: shared hit=13227 read=32228
```



PG 19 RC1

```
postgres=# alter table demo alter column value set not null
;
ALTER TABLE

postgres=# explain (verbose, analyze, buffers, costs off)
 select count(value) from demo
;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------
Aggregate (actual time=1623.554..1623.557 rows=1.00 loops=1)
   Output: count(*)
   Buffers: shared hit=2737
   ->  Index Only Scan using demo_pkey on public.demo
       (actual time=0.101..848.848 rows=1000000.00 loops=1)
         Output: key
         Heap Fetches: 0
         Index Searches: 1
         Buffers: shared hit=2737
```

```
-- Requires that the column is NOT NULL
-- works for GENERATE ALWAYS ... STORED, not for VIRTUAL
```



Why did we wait so long (PostgreSQL 19)?

You don't need it.

- The solution is easy: change COUNT(1) to COUNT(*) in your application
- ORMs can do it - Try on https://www.jooq.org/translate/ with Transform Patterns

But today, PostgreSQL is used at another level:

- enterprises pay for a hyper-scaler's managed database service
- they migrate from commercial RDBMS and expect the same optimizations
- they don't pay to be told their application is bad, and they must rewrite 20 years old code
- SQL is supposed to give this abstraction to developers: declare what you need not how to run

NOT IN or NOT EXISTS ?

PostgreSQL 18:

- SubPlan or hashed SubPlan

Transforming to an Anti Join allows more optimizations

- The query planner can optimize the join order
- Many join methods: Nested Loop, Merge Join or Hash Join

Transformation is possible when NOT NULL is guaranteed

- a NOT IN with an unknown set is never true, but an Anti Join ignores null rows

MySQL: Can transform NOT IN to an Anti Join if the row set is not null

Oracle: Can transform NOT IN to an Anti Join if the row set is not null

SQL Server: Can transform NOT IN to an Anti Join if the row set is not null

PostgreSQL: No transformation yet (PG18)

PG 18.6

```
postgres=# explain (analyze OFF, buffers, verbose, costs ON)
 select count(*) from demo
  where   key  not in (
   select key  from demo
);
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------
Aggregate  (cost=18693858641.69..18693858641.70 rows=1 width=8)
   Output: count(*)
   ->  Seq Scan on public.demo
       (cost=0.42..18693857391.67 rows=500005 width=0)
         Output: demo.key, demo.value
         Filter: (NOT (ANY (demo.key = (SubPlan 1).col1)))
         SubPlan 1
           ->  Materialize  (cost=0.42..34887.62 rows=1000010 width=8)
                 Output: demo_1.key
                 ->  Index Only Scan using demo_pkey on public.demo
                     (cost=0.42..25980.58 rows=1000010 width=8)
                       Output: demo_1.key
```

```
postgres=# explain (analyze ON, buffers, verbose, costs OFF)
 select count(*) from demo
  where   key  not in (
   select key  from demo
);
```

... still running


db<>fiddle

PG 19 RC1

```
postgres=# explain (analyze on, buffers, verbose, costs off)
 select count(*) from demo
  where   key  not in (
   select key  from demo
);
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------
Aggregate (actual time=3258.030..3258.037 rows=1.00 loops=1)
   Output: count(*)
   Buffers: shared hit=5474
   ->  Merge Anti Join (actual time=3258.022..3258.026 rows=0.00 loops=1)
         Inner Unique: true
         Merge Cond: (demo.key = demo_1.key)
         Buffers: shared hit=5474
         ->  Index Only Scan using demo_pkey on public.demo
             (actual time=0.052..819.263 rows=1000000.00 loops=1)
               Output: demo.key
               Heap Fetches: 0
               Index Searches: 1
               Buffers: shared hit=2737
         ->  Index Only Scan using demo_pkey on public.demo demo_1
             (actual time=0.030..815.955 rows=1000000.00 loops=1)
               Output: demo_1.key
               Heap Fetches: 0
               Index Searches: 1
               Buffers: shared hit=2737
```


db<>fiddle

Partial GROUP BY before JOIN (Eager aggregation)

On a star schema you want to aggregate the facts before joining to smaller dimension tables

Eager aggregation:

- pushes aggregation below joins when it is safe and cost-effective
- reduces rows participating in joins
- especially effective for fact-to-dimension joins
- can significantly reduce CPU, memory, I/O and join cost
- implemented using Partial Aggregate + Finalize Aggregate
- controlled by `enable_eager_aggregate` (default: on)

MySQL: No equivalent transformation

Oracle: Group-By Placement / Aggregation Pushdown

SQL Server: Pushdown / Partial Aggregation

PostgreSQL: No transformation before PostgreSQL 19

PG 18.6

```
postgres=# create table categories as
 select generate_series(1,10) as category_id;
postgres=# alter table categories add primary key (category_id);
postgres=# alter table demo add category_id  int default 1;
postgres=# alter table demo add foreign key (category_id)
            references categories;

postgres=# explain (analyze, buffers, verbose off, costs off)
 select category_id, avg(length(value))
 from categories join demo using (category_id)
 group by category_id
;

 QUERY PLAN
-----------------------------------------------------------------------------------------------
HashAggregate (actual time=4323.926..4323.937 rows=1.00 loops=1)
   Group Key: categories.category_id
   Batches: 1  Memory Usage: 32kB
   Buffers: shared hit=13632 read=31824
   ->  Hash Join (actual time=1.232..2721.022 rows=1000000.00 loops=1)
         Hash Cond: (demo.category_id = categories.category_id)
         Buffers: shared hit=13632 read=31824
         ->  Seq Scan on demo (actual time=0.264..1097.502 rows=1000000.00 loops=1)
               Buffers: shared hit=13631 read=31824
         ->  Hash (actual time=0.034..0.037 rows=10.00 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               Buffers: shared hit=1
               ->  Seq Scan on categories (actual time=0.012..0.021 rows=10.00 loops=1)
                     Buffers: shared hit=1
```



PG 19 RC1

```
postgres=# create table categories as
 select generate_series(1,10) as category_id;
postgres=# alter table categories add primary key (category_id);
postgres=# alter table demo add category_id  int default 1;
postgres=# alter table demo add foreign key (category_id)
            references categories;

postgres=# explain (analyze, buffers, verbose off, costs off)
 select category_id, avg(length(value))
 from categories join demo using (category_id)
 group by category_id
;

 QUERY PLAN
-----------------------------------------------------------------------------------------------
 Finalize GroupAggregate (actual time=2542.852..2542.864 rows=1.00 loops=1)
   Group Key: categories.category_id
   Buffers: shared hit=13444 read=32012
   ->  Merge Join (actual time=2542.811..2542.825 rows=1.00 loops=1)
         Merge Cond: (categories.category_id = demo.category_id)
         Buffers: shared hit=13444 read=32012
         ->  Sort (actual time=0.055..0.060 rows=2.00 loops=1)
               Sort Key: categories.category_id
               Sort Method: quicksort  Memory: 25kB
               Buffers: shared hit=1
               ->  Seq Scan on categories
                     Buffers: shared hit=1
         ->  Sort (actual time=2542.748..2542.752 rows=1.00 loops=1)
               Sort Key: demo.category_id
               Sort Method: quicksort  Memory: 25kB
               Buffers: shared hit=13443 read=32012
               ->  Partial HashAggregate
                   (actual time=2542.718..2542.722 rows=1.00 loops=1)
                     Group Key: demo.category_id
                     Batches: 1  Memory Usage: 32kB
                     Buffers: shared hit=13443 read=32012
                     ->  Seq Scan on demo (actual time=0.831..973.227 rows=1000000.00 loops=1)
                           Buffers: shared hit=13443 read=32012
```



Planner and optimizer transformations

Feature	Version/status	Authorship
Rewrite COUNT(1) and proven-non-null COUNT(expr) to COUNT(*)	PG19 core	David Rowley: CF 6186
Convert eligible NOT IN sublinks to anti joins	PG19 core	Richard Guo: CF 6455
Eager aggregation below joins	PG19 core	Richard Guo: CF 4881
Improved semi-join planning	PG19 core	Richard Guo: CF 5795
Explicit incremental sort for Append and MergeAppend	PG19 core	Richard Guo: CF 5750
Simplify IS [NOT] DISTINCT FROM for proven non-null values	PG19 core	Richard Guo: CF 6456
Earlier folding of Var IS [NOT] NULL	PG19 core	Richard Guo: CF 5685
Foreign-key-based inner-join removal	PG20-cycle proposal	Richard Guo: CF 6720
Convert ALL SubLinks to ANY SubLinks	PG20-cycle proposal	Richard Guo: CF 6627
Bloom filters for hash joins	PG20-cycle proposal	Tomas Vondra: CF 6818

Foreign Key Fast Path

The older RI (referential integrity) trigger functions used SPI to run generated SQL strings

SPI (Server Programming Interface) is PostgreSQL's internal API that lets C code build and execute SQL query strings through the full parser/planner/executor pipeline

The new feature calls lower-level storage routines directly, bypassing the query layer

- for simple non-partitioned tables
- non-temporal FK (no PERIOD clause)
- by probing the unique index directly
- a cache keeps the relation descriptor open and locked for the transaction

PG 18.6

```
postgres=# LOAD 'auto_explain';
SET auto_explain.log_min_duration = 0;
SET auto_explain.log_analyze = on;
SET auto_explain.log_nested_statements = on;
SET client_min_messages = log;

postgres=# UPDATE demo SET category_id=category_id+1 where key=42
;

LOG:  duration: 0.011 ms  plan:

Query Text: SELECT 1 FROM ONLY "public"."categories" x
WHERE "category_id" OPERATOR(pg_catalog.=) $1 FOR KEY SHARE OF x
Query Parameters: $1 = '2'

LockRows (actual time=0.009..0.010 rows=1.00 loops=1)
  ->  Seq Scan on categories x
        (actual time=0.007..0.007 rows=1.00 loops=1)
        Filter: (category_id = 2)
        Rows Removed by Filter: 3

LOG:  duration: 2.811 ms  plan:
Query Text: UPDATE demo SET category_id=category_id+1 where key=42
;

...

 Planning Time: 0.182 ms
 Trigger for constraint demo_category_id_fkey: time=0.556 calls=1
 Execution Time: 0.797 ms
```



PG 19 RC1

```
postgres=# LOAD 'auto_explain';
SET auto_explain.log_min_duration = 0;
SET auto_explain.log_analyze = on;
SET auto_explain.log_nested_statements = on;
SET client_min_messages = log;

postgres=# UPDATE demo SET category_id=category_id+1 where key=42
;

LOG:  duration: 2.808 ms  plan:
Query Text: update demo SET category_id=2 where key=42
;

...

 Planning Time: 0.132 ms
 Trigger for constraint demo_category_id_fkey: time=0.754 calls=1
 Execution Time: 1.217 ms
```



Parallel TID range scans

Some tools may read physical chunks form a heap table with CTID between ... and ...

- Change Data Capture
- Data Pipeline
- ETL
- Data Migration scripts
- Reorganization tasks

Before PostgreSQL 14 it was a SeqScan:
https://github.com/duckdb/duckdb-postgres/issues/186

After PostgreSQL 14 it was still a SeqScan if the planner choses parallel queries

PG 18.6

```
postgres=#  set max_parallel_workers_per_gather to 4;
 set parallel_tuple_cost to 0;
 set parallel_setup_cost to 0;
 set min_parallel_table_scan_size to 0;

postgres=#  explain (analyze, buffers, verbose off, costs off)
  select * from demo where ctid > '(0,0)';

 QUERY PLAN
------------------------------------------------------------------------------
 Gather (actual time=1.770..979.514 rows=1000000.00 loops=1)
   Workers Planned: 4
   Workers Launched: 4
   Buffers: shared hit=14058 read=31397
   ->  Parallel Seq Scan on demo
       (actual time=1.167..235.410 rows=200000.00 loops=5)
         Filter: (ctid > '(0,0)'::tid)
         Buffers: shared hit=14058 read=31397
 Planning Time: 0.126 ms
 Execution Time: 1749.406 ms
```



PG 19 RC1

```
postgres=#  set max_parallel_workers_per_gather to 4;
 set parallel_tuple_cost to 0;
 set parallel_setup_cost to 0;
 set min_parallel_table_scan_size to 0;

postgres=#  explain (analyze, buffers, verbose off, costs off)
  select * from demo where ctid > '(0,0)';

 QUERY PLAN
------------------------------------------------------------------------------
 Gather (actual time=1.478..978.651 rows=1000000.00 loops=1)
   Workers Planned: 4
   Workers Launched: 4
   Buffers: shared hit=83 read=45372
   ->  Parallel Tid Range Scan on demo
         (actual time=0.771..206.649 rows=200000.00 loops=5)
         TID Cond: (ctid > '(0,0)'::tid)
         Buffers: shared hit=83 read=45372
 Planning Time: 0.804 ms
 Execution Time: 1764.107 ms
(9 rows)
```



Planner and optimizer transformations

Feature	Version/status	Authorship
Direct fast paths for referential-integrity triggers	PG19 core	Amit Langote: CF
Parallel TID range scans	PG19 core	Cary Huang reviewed by David Rowley CF 5803
Faster tuple deformation	PG19 core	David Rowley: CF 6352
SIMD text and CSV COPY FROM parsing	PG19 core	https://commitfest.postgresql.org/patch/5952/
Executor batching, part 1: batch table-AM scan API	PG20-cycle proposal, Needs review in CF61	Amit Langote: CF 6176

PG 19 RC1

45 blocks were found in shared buffers and 45,410 blocks were read into them. Without read-ahead, it would have been 45,410 `pread()` calls

PostgreSQL reads ahead while scanning heap pages. Adaptive look-ahead starts small and increases when it observes I/O waits on cache misses.

- capacity is the prefetch queue's capacity (what can be pinned by the stream)
- avg and max show the average and greatest (when peak was reach) prefetch distance

The scan issued

- 2,842 I/O requests (multi block requests)
- averaging 15.98 blocks per request (45,410/ 2,842)
- waited for I/O only 8 times over 2842 requests
- in-progress=4.29 is the average number of concurrent I/O requests (so most didn't wait)

```
postgres=# explain (analyze, buffers, IO, costs off)
           select * from demo where ctid > '(0,0)'
;
                                        QUERY PLAN
-----------------------------------------------------------------------
Tid Range Scan on demo (actual time=1.014..1033.028
rows=1000000.00 loops=1)
   TID Cond: (ctid > '(0,0)'::tid)
   Prefetch: avg=65.26 max=91 capacity=94
   I/O: count=2842 waits=8 size=15.98 in-progress=4.29
   Buffers: shared hit=45 read=45410
 Planning Time: 0.187 ms
 Execution Time: 1789.778 ms
(7 rows)
```

db<>fiddle

PG 18.6

```
postgres=# \dconfig io_*

List of configuration parameters

      Parameter       | Value
----------------------+--------
 io_combine_limit     | 128kB
 io_max_combine_limit | 128kB
 io_max_concurrency   | 64
 io_method            | worker
 io_workers           | 3
```

db<>fiddle

PG 19 RC1

```
 postgres=# \dconfig io_*

  List of configuration parameters

         Parameter          | Value
----------------------------+--------
 io_combine_limit           | 128kB
 io_max_combine_limit       | 128kB
 io_max_concurrency         | 64
 io_max_workers             | 8
 io_method                  | worker
 io_min_workers             | 2
 io_worker_idle_timeout     | 1min
 io_worker_launch_interval  | 100ms
```

PG 19 RC1

```
postgres=# select pid, backend_type from pg_stat_activity where backend_type =
'io worker' order by pid;

 pid | backend_type
-----+--------------
 380 | io worker
 381 | io worker
(2 rows)

\! pgbench -iIdtg -s 10
\!  bash -c 'echo "select count(*) from pgbench_accounts where abalance >
$RANDOM" ' | pgbench -c 10 -t 100 -nf /dev/stdin &
select pid, backend_type from pg_stat_activity where backend_type = 'io
worker' order by pid
\watch

...

 pid  | backend_type
------+--------------
  380 | io worker
  381 | io worker
  414 | io worker
  415 | io worker
 1232 | io worker
 1233 | io worker
 1234 | io worker
 1449 | io worker
(8 rows)
```

db<>fiddle

PG 18.6

```
postgres=# create extension if not exists pg_buffercache;

CREATE EXTENSION

postgres=# select count(*), isdirty from pg_buffercache
            where relfilenode is not null
            group by isdirty order by 2
;
 count | isdirty
-------+---------
   739 | f
    56 | t

(2 rows)

postgres=# checkpoint;

CHECKPOINT

postgres=# select count(*), isdirty from pg_buffercache
           where relfilenode is not null
           group by isdirty order by 2
;
 count | isdirty
-------+---------
   795 | f

(1 row)
```

PG 19 RC1

```
postgres=# select count(*), isdirty from pg_buffercache
           where relfilenode is not null
           group by isdirty order by 2
;
 count | isdirty
-------+---------
   795 | f

(1 row)


postgres=# select * from pg_buffercache_mark_dirty_all();

 buffers_dirtied | buffers_already_dirty | buffers_skipped
-----------------+-----------------------+-----------------
             794 |                     1 |               0

(1 row)

postgres=# select count(*), isdirty from pg_buffercache
           where relfilenode is not null
           group by isdirty order by 2
;
 count | isdirty
-------+---------
   795 | t

(1 row)
```

db<>fiddle

Shared memory, I/O, WAL, and observability

Feature	Version/status	Authorship
EXPLAIN (ANALYZE, IO)	PG19 core	Tomas Vondra 681daed93
auto_explain.log_io	PG19 core	Tomas Vondra 61c36a34a
Automatic sizing of the worker I/O pool	PG19 core	Thomas Munro: CF 5913
AIO read-ahead scheduling for large requests	PG19 core	Andres Freund a9ee66881 8ca147d58 f63ca3379
Do not wait synchronously for already-running read-stream I/O	PG19 core	Andres Freund: CF 6127
Simplified shared-memory registration with ShmemRequestStruct()	PG19 core	Ashutosh Bapat and Heikki Linnakangas: CF 6443
Reduce VACUUM and COPY FREEZE WAL by eliminating xl_heap_visible	PG19 committed	Melanie Plageman: CF 5841
Mark pg_buffercache buffers dirty	PG19 core	Nazir Bilal Yavuz; release commit 9ccc049df
Resizable shared-memory structures	PG20-cycle proposal	Ashutosh Bapat: CF 6652
Index prefetching	PG20-cycle proposal	Peter Geoghegan and Tomas Vondra: CF 4351
Basic NUMA awareness	PG20-cycle proposal	Tomas Vondra: CF 6049
Backward I/O combining	PG20-cycle proposal	Nazir Bilal Yavuz: CF 7035
Async fsync for recovery and checkpointer	PG20-cycle proposal	Nazir Bilal Yavuz: CF 7190
Checkpointer write combining	PG20-cycle proposal	Melanie Plageman: CF 7171
Preallocate WAL segments	PG20-cycle proposal	Ayush Tiwari: CF 7052
Batch fsyncs while recycling WAL segments	PG20-cycle proposal	Ayush Tiwari: CF 7088

PG 18.6

The databases must be closed in order to calculate checksums

PG 20 devel

```
postgres=# select pg_enable_data_checksums();
postgres=# show data_checksums;

 data_checksums
----------------
 inprogress-on

postgres=# show data_checksums;

 data_checksums
----------------
 on

-- corrupt the file

postgres=# select * from pg_buffercache_evict_all();

postgres=# \! cat -v $file | tail -c 42
@^@^@^@^A^@^A^@^B       ^X^@^[Hello World!^@^@^@

postgres=# \! LC_ALL=C sed 's/World!/Hacker/g' $file > /tmp/corrupted.file &&
cat /tmp/corrupted.file > $file

postgres=# \! cat -v $file | tail -c 42
@^@^@^@^A^@^A^@^B       ^X^@^[Hello Hacker^@^@^@postgres=#


postgres=# select distinct value from hackme;
ERROR:  invalid page in block 0 of relation "base/5/24627"
```



Online operations, TLS, recovery, and correctness

Feature	Version/status	Authorship
Enable and disable data checksums online	PG19 reverted	Daniel Gustafsson and Magnus Hagander: CF 5323
Handle invalid or dropped databases during checksum enable	Committed follow-up	Ayush Tiwari: CF 7087
Server-side SNI	PG19 core	Daniel Gustafsson and Jacob Champion: CF 4984
Thread-safe libpq OpenSSL support	PG19 committed	Daniel Gustafsson: CF 5868
debug_exec_backend	PG19 core	Daniel Gustafsson; release commit b3fe098d3
Remove RADIUS authentication	PG19 core	Thomas Munro: CF 6474
Remove MULE_INTERNAL encoding	PG19 core	Thomas Munro: CF 6472
Fix crash during recovery when a redo segment is missing	Committed	Nitin Jadhav: CF 6279
Send nontransactional invalidations for in-place updates	Committed	Noah Misch: CF 5091

PG 18.6

```
postgres=# insert into demo values(1000042,'Hi');
INSERT 1

postgres=# explain (analyze, wal, costs off)
           select key from demo where key=1000042
;
                                    QUERY PLAN
-------------------------------------------------------------------------
 Index Only Scan using demo_pkey on demo
   (actual time=0.048..0.053 rows=1.00 loops=1)
   Index Cond: (key = 1000042)
   Heap Fetches: 1
   Index Searches: 1
   Buffers: shared hit=5

postgres=# explain (analyze, wal, costs off)
           select key from demo where key=1000042
;
                                    QUERY PLAN
-------------------------------------------------------------------------
 Index Only Scan using demo_pkey on demo
   (actual time=0.033..0.037 rows=1.00 loops=1)
   Index Cond: (key = 1000042)
   Heap Fetches: 1
   Index Searches: 1
   Buffers: shared hit=5
```



PG 19 RC1

```
postgres=# insert into demo values(1000042,'Hi');
INSERT 1

postgres=# explain (analyze, wal, costs off)
           select key from demo where key=1000042
;
                                    QUERY PLAN
-------------------------------------------------------------------------
Index Only Scan using demo_pkey on demo
   (actual time=0.024..0.026 rows=1.00 loops=1)
   Index Cond: (key = 1000042)
   Heap Fetches: 1
   Index Searches: 1
   Buffers: shared hit=7
   WAL: records=1 bytes=60

postgres=# explain (analyze, wal, costs off)
           select key from demo where key=1000042
;
                                    QUERY PLAN
-------------------------------------------------------------------------
 Index Only Scan using demo_pkey on demo
   (actual time=0.038..0.042 rows=1.00 loops=1)
   Index Cond: (key = 1000042)
   Heap Fetches: 0
   Index Searches: 1
   Buffers: shared hit=4
```

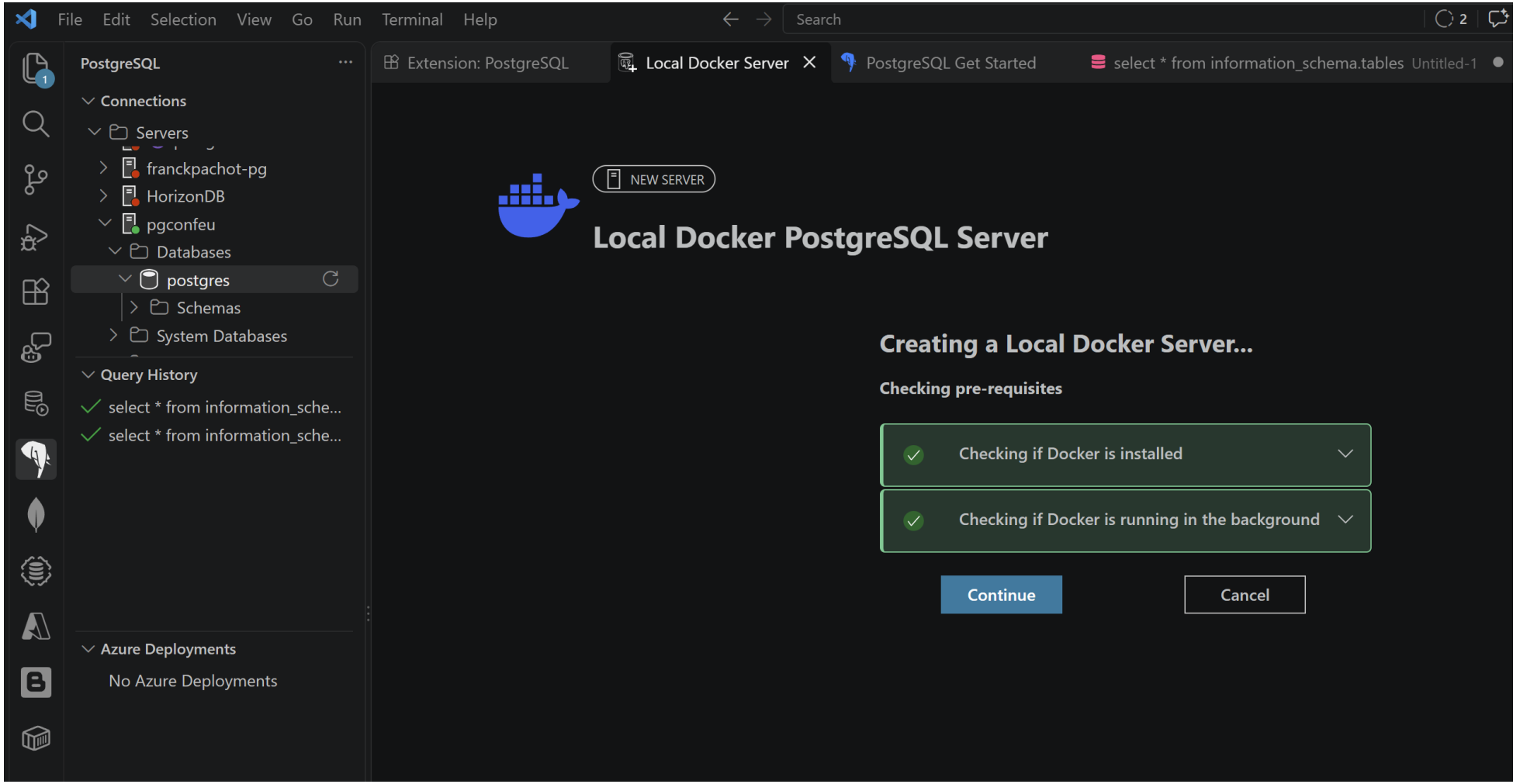


Online operations, TLS, recovery, and correctness

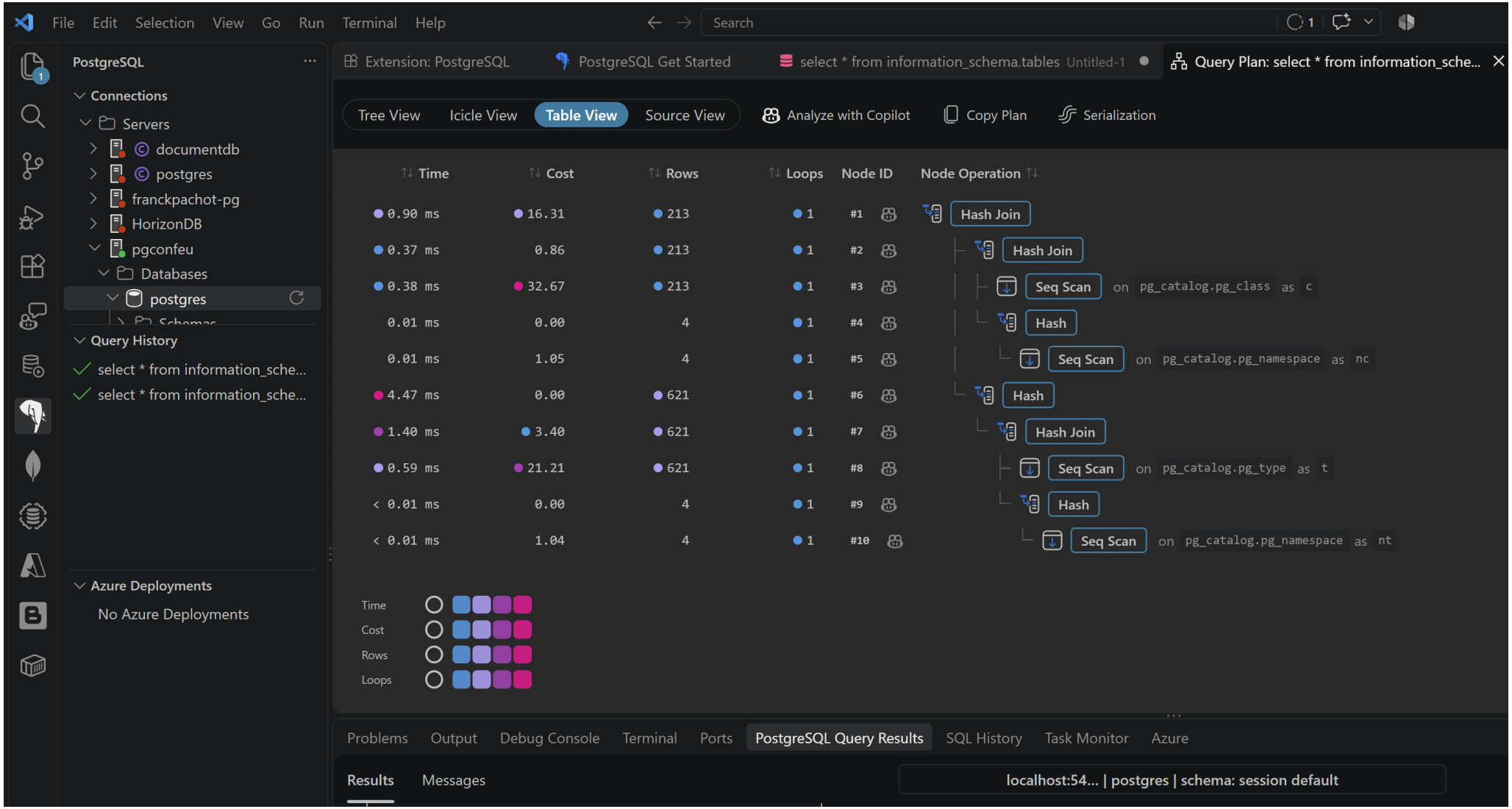
Feature	Version/status	Authorship
Reads maintain visibility-map state	PG19 core	Melanie Plageman b46e1e54d
Reduce EXPLAIN ANALYZE timing overhead with rdtsc	PG19 core	Andres Freund, Lukas Fittl, and David Geier: CF 5622
SQL/JSON JSON_TABLE PLAN clause	PG20-cycle proposal	Amit Langote: CF 5478
Correct JSON_ARRAY(query) empty-input behavior	PG19 core	Richard Guo: CF 6628
Publish all sequences in logical replication	PG19 core	Tomas Vondra and Vignesh C; release commit 96b378497
Rename the BUFFERPIN wait event	PG19 core	Andres Freund; release commit 6c5c393b7
SQL Property Graph Queries	PG19 reverted -> PG20?	Peter Eisentraut and Ashutosh Bapat: CF 4904
SQL/PGQ error and assertion hardening		Ayush Tiwari: CF 6742 and CF 6786
INSERT ... BY NAME	PG20-cycle proposal	Ayush Tiwari: CF 6982
Cache partition-index parent OIDs	PG20-cycle proposal	Ayush Tiwari: CF 7030
		Richard Guo with Dmitry Koval, Alexander

PostgreSQL for Visual Studio Code

https://marketplace.visualstudio.com/items?itemName=ms-ossdata.vscode-pgsql



PostgreSQL for Visual Studio Code





Website · Docs · Quick Example · GitHub



Blog post:

Automate Postgres-to-Blob with durable workflow (pg_durable + pg_duckdb)

https://dev.to/franckpachot/automate-the-postgres-to-blob-archive-with-a-durable-workflow-pgdurable-pgduckdb-2edc

```
SELECT df.start(
  -- 1. count the cold slice, capture it as $n
    $Step1$
       SELECT count(*) AS n FROM orders WHERE order_date < DATE '2024-01-01'
    $Step1$ |=> 'n'
  -- 2. write those rows to Blob as Parquet, through the DuckDB engine
  ~> $Step2$
       SELECT duckdb.raw_query(
       $DuckDB$
          COPY (
            SELECT order_id, order_date, amount
            FROM pgduckdb.public.orders
            WHERE order_date < DATE '2024-01-01'
          ) TO 'az://lake/archive/orders_2023.parquet' (FORMAT parquet)
       $DuckDB$
       )
    $Step2$
  -- 3. verify the file reads back, capture the row count as $v
  ~> $Step3$
       SELECT count(*) AS v
       FROM read_parquet('az://lake/archive/orders_2023.parquet')
    $Step3$ |=> 'v'
  -- 4. delete only if the archive count matches
  ~> ( $Check$ SELECT $v = $n AS archive_complete $Check$
     ?> $Del$  DELETE FROM orders WHERE order_date < DATE '2024-01-01' $Del$
     !> $Fail$ DO $do$ BEGIN
                 RAISE EXCEPTION 'archive incomplete: expected % rows, file ha
               END $do$;
             $Fail$
    )
  -- 5. record what happened
  ~> $Step5$
       INSERT INTO archive_log(archived_year, n_rows, blob)
       VALUES (2023, $n, 'az://lake/archive/orders_2023.parquet')
    $Step5$,
  'durable-archive'    -- a label to find the run by
);
```



Website · Docs · Quick Example · GitHub



- Long-running, fault-tolerant SQL
- Replaces cron + workers + queues + status tables
- Can replace DBOS or Temporal if your workflow's center of gravity is completely inside the DB

Examples:

- Vector embedding pipelines: chunk, call an embedding API, and upsert into pgvector.
- Ingest pipelines: stage, deduplicate, transform, and publish large batches
- Scheduled maintenance: detect bloat, notify, wait for approval, then run the next action
- Fan-out aggregation: run independent queries in parallel, then join the results

DocumentDB



```
# podman run -d --replace --name PGDD -e POSTGRES_PASSWORD=x \
     ghcr.io/documentdb/documentdb/documentdb-local:latest \
     --username franck --password franck --start-pg

# podman exec -it PGDD mongosh -u franck -p franck
'mongodb://localhost:10260/?tls=true&tlsAllowInvalidCertificates=true'

test> db.demo.createIndex({ "a": 1, "ts": -1 });

test> for (let i = 0; i < 10000; i++) {
  db.demo.insertOne({ a: 1, b: Math.random(), ts: new Date() });
}

db.demo.find( { a: 1 } ).sort( { ts: -1 } ).limit(10).explain();
...
 winningPlan: {
      stage: 'LIMIT',
      inputStage: {
        stage: 'FETCH', ns: 'test.demo',
        inputStage: {
          stage: 'IXSCAN', ns: 'test.demo',
          indexName: 'a_1_ts_-1', direction: 'Forward',
          indexUsage: {
            indexKeyString: '{"a": 1,"ts": -1}',
            isMultiKey: false,
            bounds: [ '["a": [1, 1], "ts": DESC(MinKey, MaxKey)]' ]
          },
          hasOrderBy: true, indexFilterSet: [ { a: { '$eq': 1 } } ],
        }
      }
...
```

```
# podman exec -it PGDD psql -p 9712 postgres -U franck

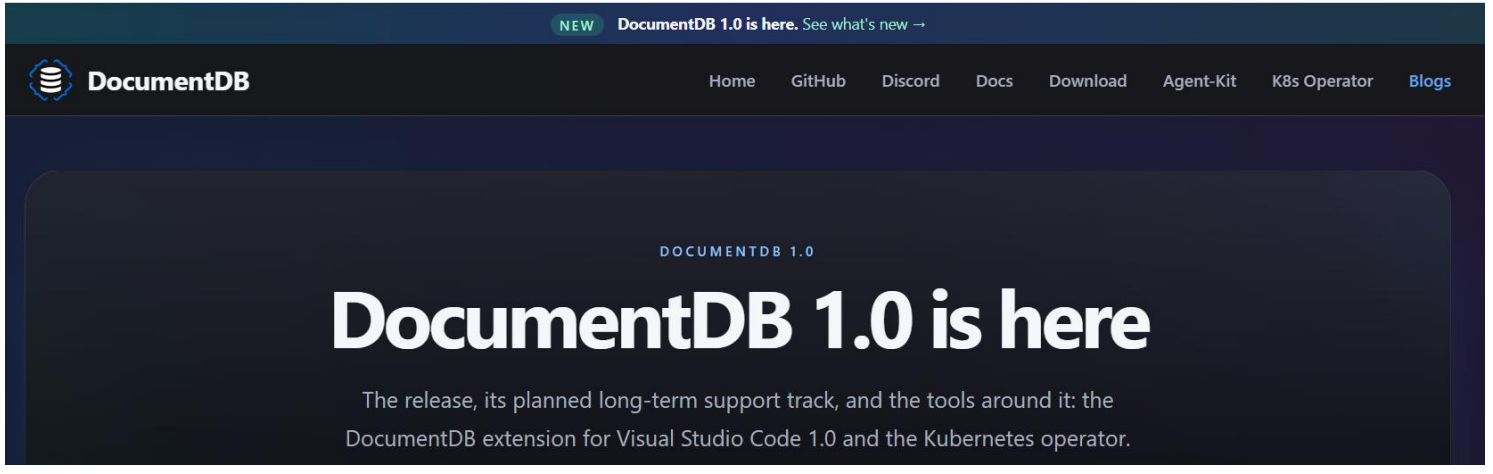
explain (analyze, buffers, verbose, costs off)
select document from bson_aggregation_find(
  'test', '{
    "find": "demo",
    "filter": { "a": 1 },
    "sort":   { "ts": -1 },
    "limit": 10
  }'::documentdb_core.bson
);

...
 -> Index Scan using "a_1_ts_-1" on documentdb_data.documents_2 collection
   (actual time=0.056..0.067 rows=10 loops=1)
    Output: document
    Index Cond: (collection.document @= '{ "a": {"$numberInt":"1"} }'::bson)
    Order By: (collection.document |-<> '{ "ts": {"$numberInt":"-1"} }'::bson)
    Buffers: shared hit=5
...
     Column      |  Type  | Collation | Nullable | Default
-----------------+--------+-----------+----------+---------
 shard_key_value | bigint |           | not null |
 object_id       | bson   |           | not null |
 document        | bson   |           | not null |
Indexes:
    "collection_pk_2" PRIMARY KEY, btree (shard_key_value, object_id)
    "documents_rum_index_3" documentdb_extended_rum (document
documentdb_extended_rum_catalog.bson_extended_rum_composite_path_ops
(pathspec='[ "a", { "ts" : -1 } ]', tl='2691'))
```

db<>fiddle

Extensions and the PostgreSQL ecosystem

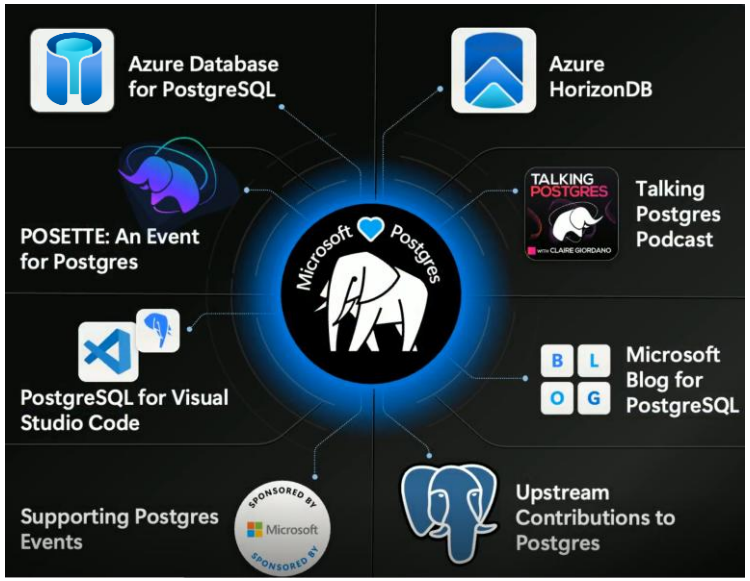
Project	Version/status	Attribution
PG_Layers pglayers-azure	PG 17, 18, 19 (beta) PG>=18 uses extension_control_path	Ismael Mejia
DocumentDB	Pre-GA PG_Layers contains 0.114-0 db<>fiddle contains 0.116-0	Microsoft created and contributed the project to the Linux Foundation
pg_durable	Preview extension for PostgreSQL 17 and 18	Microsoft open-source project under the PostgreSQL License
Microsoft PostgreSQL extension for VS Code	Available for VS Code, Cursor, Kiro	



Postgres 19, 20, & Beyond

Hyperscalers contribute to PostgreSQL

- Employ full-time developers and committers Developers,
- Release extensions to improve developer experience
- Sponsor and organize conferences
- but most importantly:
  feedback from critical enterprise workloads



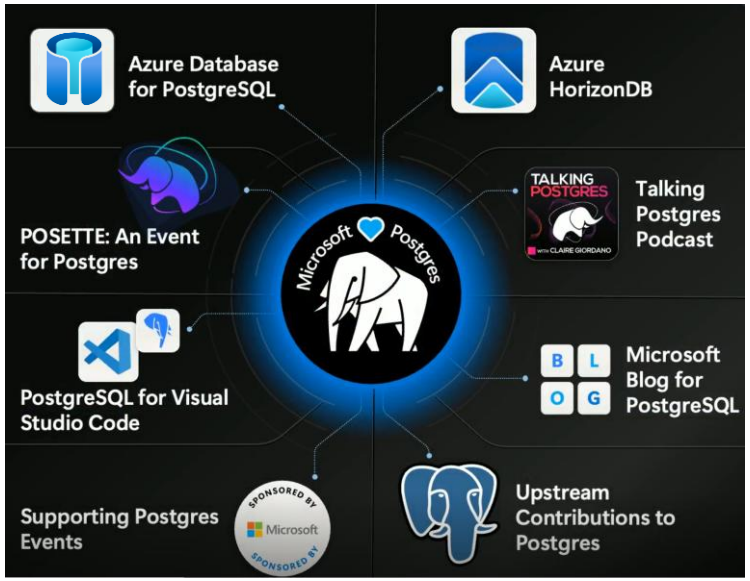
PostgreSQL popularity requires lots of effort

- PG19: four beta released, two release candidates planned
  (https://wiki.postgresql.org/wiki/PostgreSQL_19_Open_Items)

Microsoft leverages the ecosystem: support community extensions in the managed services with minimal changes (e.g `pg_duckdb` on Azure: https://dev.to/franckpachot/series/44985)

Microsoft Azure

Thank you





Franck Pachot
Developer Advocate
Principal Product Manager