PostgreSQL 16 and beyond

Amit Kapila

PostgreSQL Committer

and Major Contributor

FUJITSU

Evolution of the OSS database PostgreSQL

FUJITSU

- Ongoing version upgrades once a year
- Enhanced support for large volume data in recent years

6.0 — 1997

First PostgreSQL Release
Inherits Postgres project at the University of California, Berkeley, which had been running since 1986

9.0 — 2010

- Streaming Replication

9.1 — 2011

- Synchronous Replication
- Foreign tables
- Unlogged tables

9.2 — 2012

- Index-only scans
- json data type

9.3 — 2013

- Update to Foreign Data Wrapper
- Materialized views

9.4 — 2014

- jsonb data type

9.5 — 2016 (Jan)

- Row-level security
- BRIN Index

9.6 — 2016 (Sep)

- Parallel Sequential Scan
- Multiple standby servers in sync rep

10 — 2017

- Declarative Partitioning
- Logical Replication

11 — 2018

- SQL stored procedures
- Partitioning by hash key

12 — 2019

- Partitioning performance enhancements
- Table Access Methods

13 — 2020

- De-duplication in B-tree index
- Incremental sorting
- Parallelized vacuum for indexes

14 — 2021

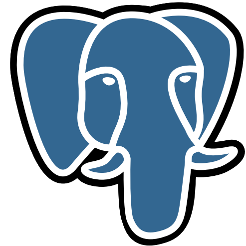
- Snapshot scalability (better reads)
- Logical Replication for in-progress txns
- Reduce bloat for B-tree index updates
- Parallel foreign table scans via postgres_fdw

15 — 2022

- Merge command
- Shared memory stats
- Row/Column Filtering in Logical Replication
- Server-side compression for backups

16 — 2023 (Q3)

- Logical replication from standby
- Parallel apply of large txns
- SQL/JSON standard compliance
- Faster relation extension
- Load balancing via libpq



© Fujitsu 2023

Agenda

FUJITSU

- Key features and performance improvements in PostgreSQL 16
- PostgreSQL 17 and beyond

Agenda

FUJITSU

- Key features and performance improvements in PostgreSQL 16
- PostgreSQL 17 and beyond

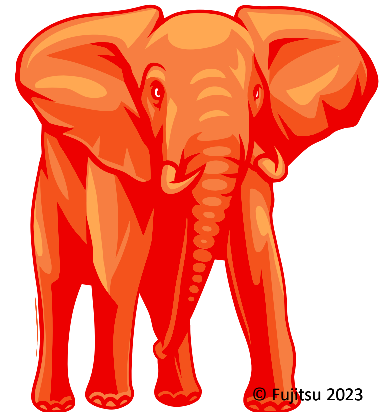
Logical replication improvements

- Allows to filter the data based on origin during replication

```
CREATE PUBLICATION  mypub FOR ALL TABLES;
CREATE SUBSCRIPTION mysub CONNECTION 'dbname=postgres'
       PUBLICATION  mypub WITH (origin = none);
```

  - This can be used to setup n-way logical replication and that will prevent loops when doing bi-directional replication

- Allow logical decoding from standby
  - This requires wal_level = logical on both primary and standby
  - This can be used for workload distribution by allowing subscribers to subscribe from standby when primary is busy



© Fujitsu 2023

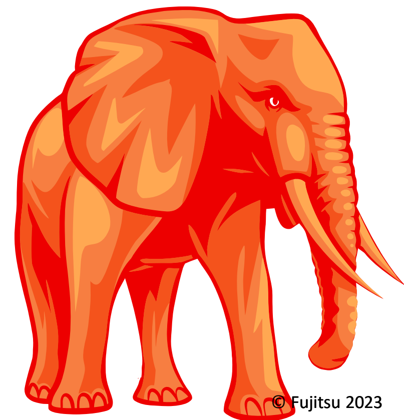
Logical replication improvements

FUJITSU

- Allow apply process to perform operations with the table owner's privileges

```
CREATE SUBSCRIPTION mysub CONNECTION ...
       PUBLICATION mypub WITH (run_as_owner = false);
```

- Allow non-superusers to create subscription
  - The non-superusers must have been granted pg_create_subscription role
  - The non-superusers are required to specify a password for authentication
  - The superusers can set password_required = false for non-superusers that own the subscription

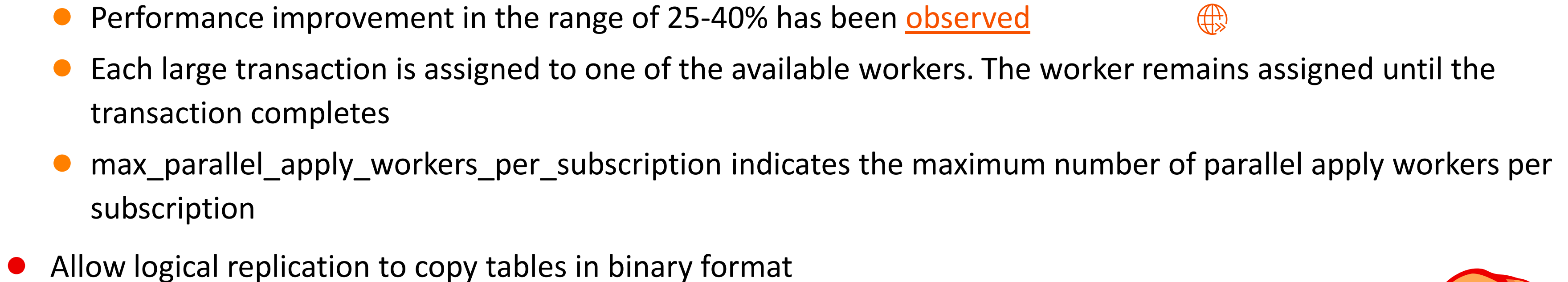


© Fujitsu 2023

Logical replication improvements

- Allow the large transactions to be applied in parallel

```
CREATE SUBSCRIPTION mysub CONNECTION ...
       PUBLICATION mypub WITH (streaming = parallel);
```

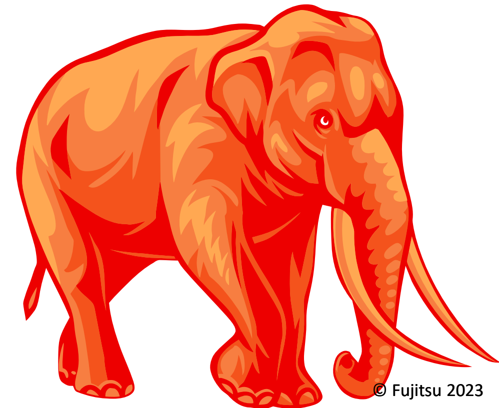


  - Performance improvement in the range of 25-40% has been observed
  - Each large transaction is assigned to one of the available workers. The worker remains assigned until the transaction completes
  - max_parallel_apply_workers_per_subscription indicates the maximum number of parallel apply workers per subscription

- Allow logical replication to copy tables in binary format

```
CREATE SUBSCRIPTION mysub CONNECTION ...
       PUBLICATION mypub WITH (binary = true);
```

  - Copying tables in binary format may reduce the time spent, depending on column types

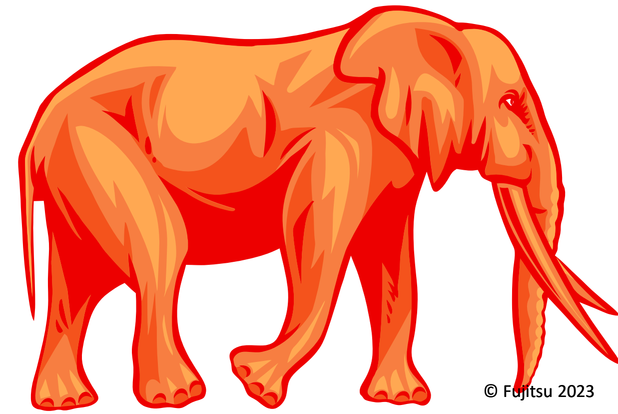


© Fujitsu 2023

Logical replication improvements

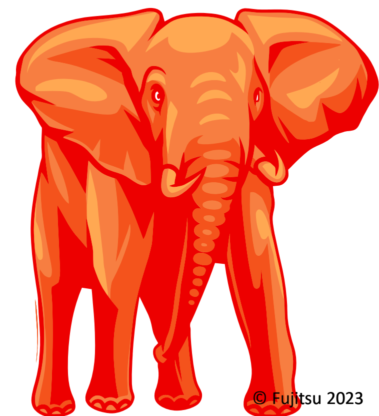
FUJITSU

- Allow the use of indexes other than PK and REPLICA IDENTITY on the subscriber
  - Using REPLICA IDENTITY FULL on the publisher can lead to a full table scan per tuple change on the subscriber when REPLICA IDENTITY or PK index is not available
  - The index that can be used must be a btree index, not a partial index, and it must have at least one column reference
  - The performance improvement is proportional to the amount of data in the table



© Fujitsu 2023

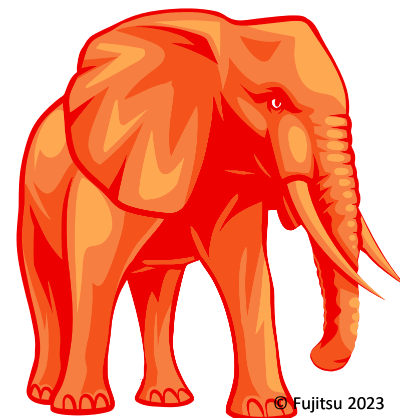
- Faster relation extension
  - Provides significant improvement (3X for 16 clients) for concurrent COPY into a single relation
  - Previously, while holding the relation extension lock, we used to:
    - Acquiring a victim buffer for the new page. This may further require writing out the old page contents including possibly needing to flush WAL
    - We write a zero page during the extension, and then later write out the actual page contents. This can nearly double the write rate
  - Now, the relation extension lock is held just for extending the relation
- Allow HOT updates if only BRIN-indexed columns are updated
  - We still update BRIN-index if the corresponding columns are updated
  - This does not apply to attributes referenced in index predicates, an update of such attribute always disables HOT



© Fujitsu 2023

FUJITSU

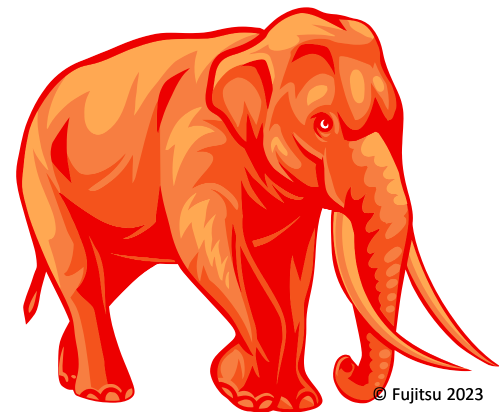
- Direct I/O
  - This allows to ask the kernel to minimize caching effects for relation data and WAL files
  - Currently this feature reduces performance and is not intended for end users, so disabled by default
  - Can enable by GUC debug_io_direct
    - Valid values: data, wal, wal_init
  - The further plan is to introduce our own I/O mechanisms, read-ahead, etc. to replace the facilities the kernel disables with this option.
  - Align all I/O buffers at 4096 to have a better performance with direct I/O
- Allow freezing at page level during vacuum
  - This reduces the cost of freezing by reducing WAL volume



© Fujitsu 2023

Storage (3/4)

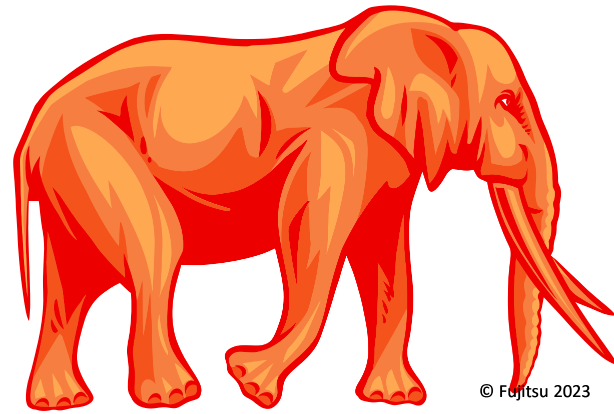
- pg_stat_io view to show detailed I/O statistics
  - It contains one row for each combination of backend type, target I/O object, and I/O context, showing cluster-wide I/O statistics
  - Example of backend types: background worker, autovacuum worker, checkpointer, etc.
  - Possible type of target I/O objects: Permanent or Temporary relations
  - Possible values of I/O context: normal, vacuum, bulkread, bulkwrite
  - It tracks various I/O operations like reads, writes, extends, hits, evictions, reuses, fsyncs
  - A high evictions count can indicate that shared buffers should be increased
  - Large numbers of fsyncs by client backends could indicate misconfiguration of shared buffers or misconfiguration of the checkpointer
  - The stats doesn't differentiate between data which had to be fetched from disk and that which already resided in the kernel page cache



© Fujitsu 2023

FUJITSU

- Allow Vacuum/Analyze to specify buffer usage limit
  - A new option BUFFER_USAGE_LIMIT has been added
  - This allows user to control the size of shared buffers to use
  - Larger values can make vacuum run faster at the cost of slowing down other concurrent queries
  - vacuum_buffer_usage_limit (GUC) allows another way to control but BUFFER_USAGE_LIMIT would take precedence
  - GUC allows even autovacuum to use the specified limit
  - Add --buffer-usage-limit option to vacuumdb



© Fujitsu 2023

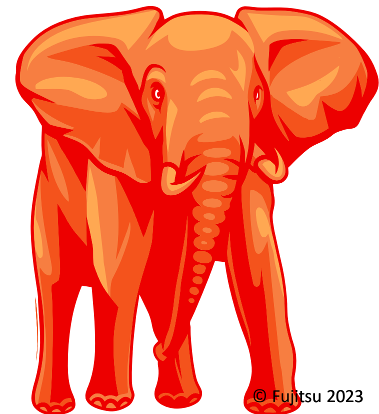
SQL

FUJITSU

- Improves general support for text collations, which provide rules for how text is sorted

```
CREATE COLLATION en_custom (provider = icu, locale = 'en', rules = '&a < g');
```

  - This places g fter a efore b
  - See specifications for details
  - New options are added to CREATE COLLATION, CREATE DATABASE, createdb, and initdb to set the rules
- Allows ICU to be the default collation provider
  - The decision to make it default is still under discussion
- Adds support for the predefined Unicode and ucs_basic collations



© Fujitsu 2023

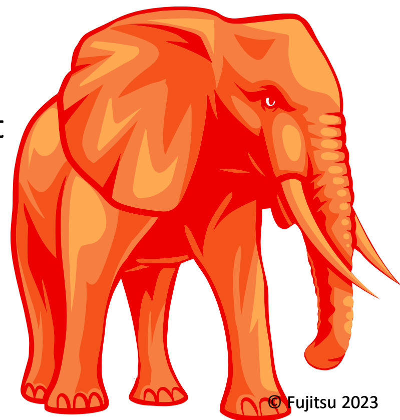
- SQL/JSON standard-conforming constructors for JSON types

JSON_ARRAY() Constructs a JSON array from either a series of value_expression parameters or from the results of query_expression

JSON_ARRAYAGG() Behaves in the same way as json_array but as an aggregate function so it only takes one value_expression parameter

JSON_OBJECT() Constructs a JSON object of all the key/value pairs given, or an empty object if none are given

JSON_OBJECTAGG() Behaves like json_object, but as an aggregate function, so it only takes one key_expression and one value_expression parameter



© Fujitsu 2023

- SQL/JSON standard-conforming constructors for JSON types

JSON_ARRAY()

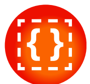
```
SELECT json_array(1,true,json '{"a":null}');
      json_array
-----------------------
 [1, true, {"a":null}]
```

JSON_ARRAYAGG()

```
SELECT json_arrayagg(v NULL ON NULL) FROM (VALUES(2),(1),(3),(NULL)) t(v);
  json_arrayagg
-----------------
 [2, 1, 3, null]
```

JSON_OBJECT()

```
SELECT json_object('code' VALUE 'P123', 'title': 'Jaws', 'title1' : NULL ABSENT ON NULL);
             json_object
-------------------------------------
 {"code" : "P123", "title" : "Jaws"}
```



JSON_OBJECTAGG()

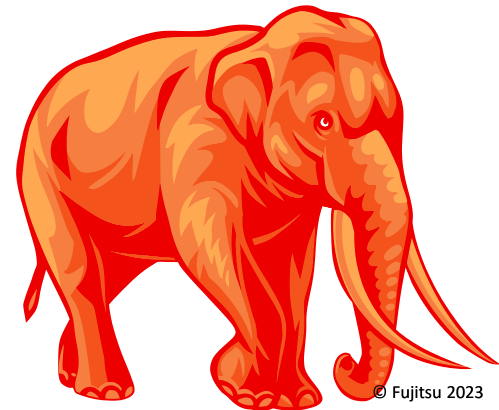
```
SELECT json_objectagg(k:v) FROM (VALUES ('a'::text,current_date),('b',current_date + 1)) AS t(k,v);
               json_objectagg
--------------------------------------------
 { "a" : "2023-05-19", "b" : "2023-05-20" }
```


© Fujitsu 2023

FUJITSU

- Introduce SQL standard IS JSON predicate
  - IS JSON [VALUE]
  - IS JSON ARRAY
  - IS JSON OBJECT
  - IS JSON SCALAR

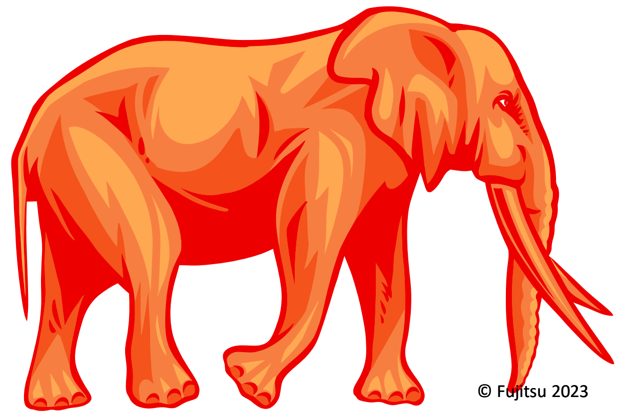
```
SELECT js, js IS JSON "json?", js IS JSON SCALAR "scalar?",
       js IS JSON OBJECT "object?", js IS JSON ARRAY "array?"
FROM (VALUES ('123'), ('"abc"'), ('{"a": "b"}'), ('[1,2]')) foo(js);

     js      | json? | scalar? | object? | array?
------------+-------+---------+---------+--------
 123         | t     | t       | f       | f
 "abc"       | t     | t       | f       | f
 {"a": "b"} | t     | f       | t       | f
 [1,2]       | t     | f       | f       | t
```



© Fujitsu 2023

- Parallel Hash Full Join

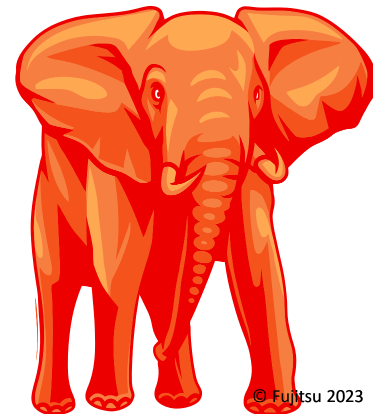
```
EXPLAIN (COSTS OFF)
  SELECT  COUNT(*)
  FROM simple r FULL OUTER JOIN simple s USING (id);
                        QUERY PLAN
-------------------------------------------------------------
 Finalize Aggregate
   ->  Gather
         Workers Planned: 2
         ->  Partial Aggregate
               ->  Parallel Hash Full Join
                     Hash Cond: (r.id = s.id)
                     ->  Parallel Seq Scan on simple r
                     ->  Parallel Hash
                           ->  Parallel Seq Scan on simple s
```



© Fujitsu 2023

- Allow parallel aggregate on string_agg and array_agg

```
EXPLAIN (COSTS OFF)
  SELECT y, string_agg(x::text, ',') AS t, array_agg(x) AS a
  FROM pagg_TEST GROUP BY y;
                    QUERY PLAN
--------------------------------------------------
 Finalize HashAggregate
   Group Key: y
   ->  Gather
         Workers Planned: 2
         ->  Partial HashAggregate
               Group Key: y
               ->  Parallel Seq Scan on pagg_test
```

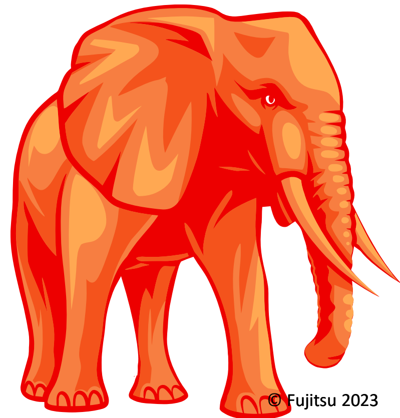


© Fujitsu 2023

FUJITSU

- Allow aggregates having ORDER BY or DISTINCT to use pre-sorted data
  - Previously, we always needed to sort tuples before doing aggregation
  - Now, an index could provide pre-sorted input which will be directly used for aggregation

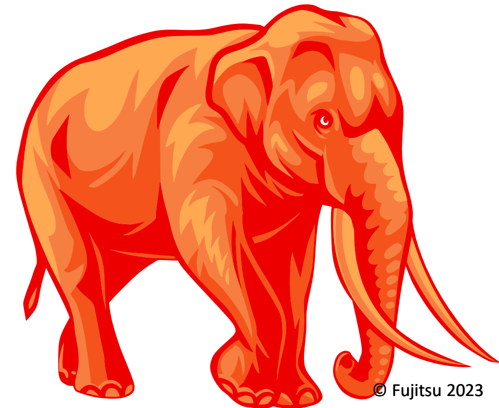
```
EXPLAIN (COSTS OFF)
  SELECT SUM(c1 order by c1), MAX(c2 ORDER BY c2) FROM presort_test;
                         QUERY PLAN
------------------------------------------------------------
 Aggregate
   ->  Index Scan using presort_test_c1_idx on presort_test

SET enable_presorted_aggregate=off;

EXPLAIN (COSTS OFF)
  SELECT SUM(c1 ORDER BY c1), MAX(c2 ORDER By c2) FROM presort_test;
           QUERY PLAN
--------------------------------
 Aggregate
   ->  Seq Scan on presort_test
```



© Fujitsu 2023

- Cache the last found partition for RANGE and LIST partition lookups
  - This reduces the overhead of bulk-loading into partitioned tables where many consecutive tuples belong to the same partition
- Allow left join removals and unique joins on partitioned tables

```
CREATE TEMP TABLE a (id int PRIMARY KEY, b_id int);
CREATE TEMP TABLE parted_b (id int PRIMARY KEY) PARTITION BY RANGE(id);
CREATE TEMP TABLE parted_b1 PARTITION OF parted_b FOR VALUES FROM (0) TO (10);

EXPLAIN (COSTS OFF)
  SELECT a.* FROM a LEFT JOIN parted_b pb ON a.b_id = pb.id;
  QUERY PLAN
---------------
 Seq Scan on a
```

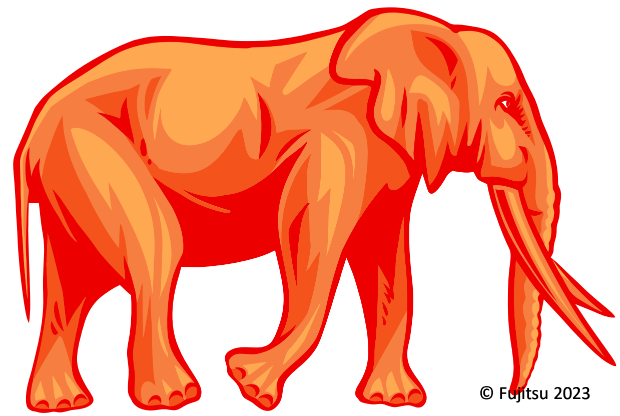


© Fujitsu 2023

Security/privileges

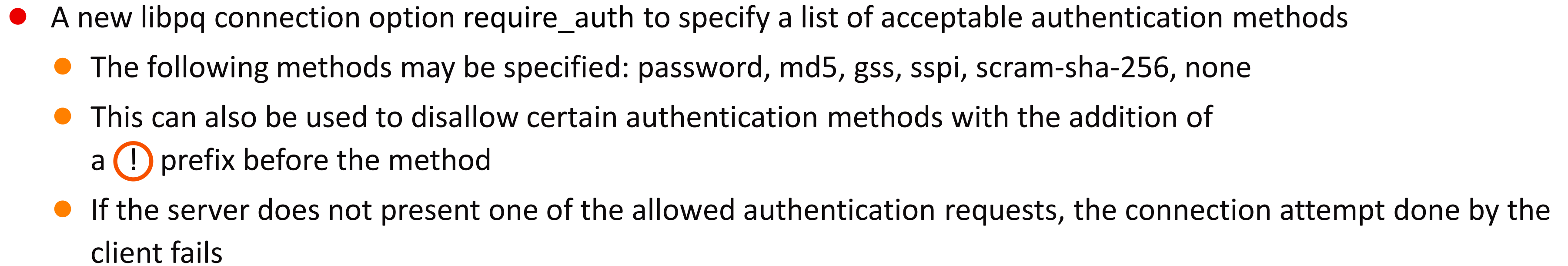
FUJITSU

- Avoid the need to grant superuser privileges for following
  - pg_maintain allows executing VACUUM, ANALYZE, CLUSTER, REFRESH MATERIALIZED VIEW, REINDEX, and LOCK TABLE on all relations
  - Alternatively, one can grant MAINTAIN privilege to users
  - reserved_connections provides a way to reserve connection slots for non-superusers
  - pg_use_reserved_connections allows the use of connection slots reserved via reserved_connections
- Add support for Kerberos credential delegation
  - This allows the PostgreSQL server to then use those delegated credentials to connect to another service, such as with postgres_fdw or dblink or theoretically any other service which is able to be authenticated using Kerberos

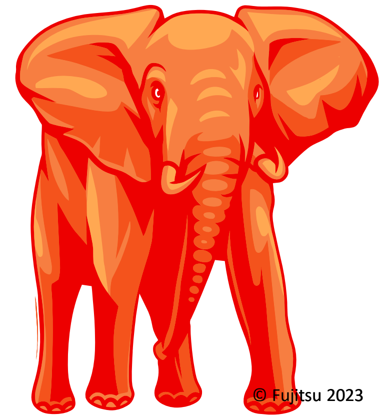


© Fujitsu 2023

Security/privileges



- A new libpq connection option require_auth to specify a list of acceptable authentication methods
  - The following methods may be specified: password, md5, gss, sspi, scram-sha-256, none
  - This can also be used to disallow certain authentication methods with the addition of a ! prefix before the method
  - If the server does not present one of the allowed authentication requests, the connection attempt done by the client fails



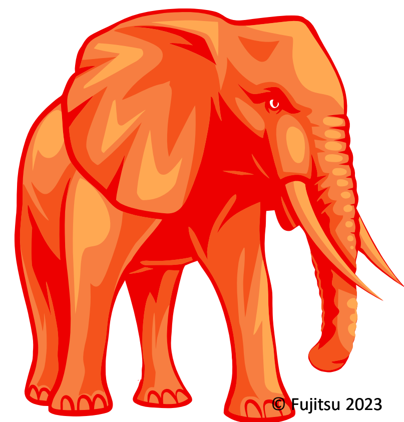
Fujitsu-Public

© Fujitsu 2023

Security/privileges

FUJITSU

- Introduce GRANT ... SET option
  - The SET option, if it is set to TRUE, allows the member to change to the granted role using the SET ROLE command
  - To create an object owned by another role or give ownership of an existing object to another role, you must have the ability to SET ROLE to that role
  - Otherwise, commands such as ALTER ... OWNER TO or CREATE DATABASE ... OWNER will fail

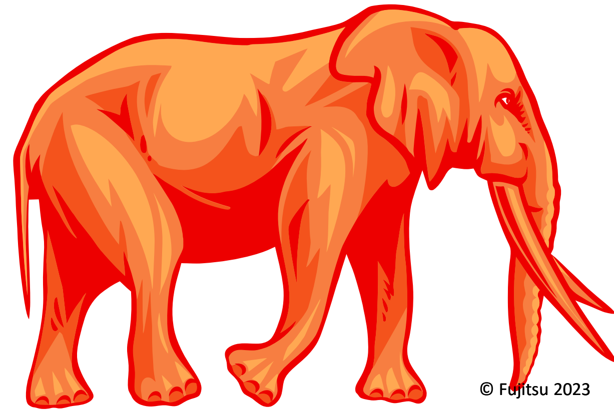


© Fujitsu 2023

Miscellaneous performance improvements

FUJITSU

- Support for CPU acceleration using SIMD for both x86 and ARM architectures
  - Optimizations for processing ASCII and JSON strings, and subtransaction searches
- Connection load balancing in libpq
  - load_balance_hosts = random allows hosts and addresses will be connected to in random order
  - This parameter can be used in combination with target_session_attrs to load balance over standby servers only
  - It is recommended to also configure a reasonable value for connect_timeout to allow other nodes to be tried when the chosen one is not responding
- Added LZ4 and Zstandard compression options to pg_dump
- Allow COPY into foreign tables to add rows in batches
  - This is controlled by the postgres_fdw batch_size option

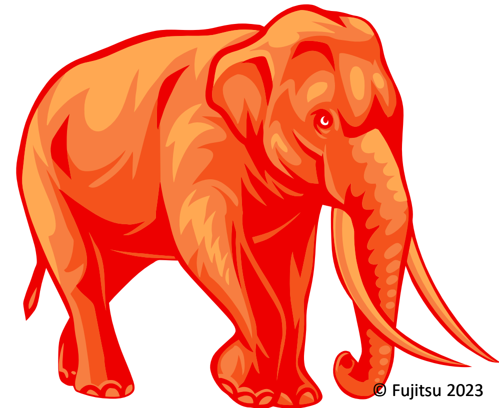


© Fujitsu 2023

Miscellaneous performance improvements

FUJITSU

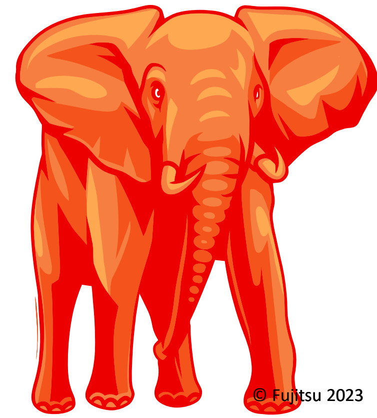
- Improve performance of pg_strtointNN functions
  - Testing has shown about 8% speedup of COPY into a table containing 2 INT columns
- Improve speed of hash index builds
  - In initial data sort, if the bucket numbers are the same then next sort on hash value
  - Speedup hash index builds by skipping needless binary searches
  - Hash Index build speed up by 5-15%
- Improve performance of and reduce overheads of memory management
  - Reduce the header size for each allocation from 16 or more bytes to 8 bytes
  - Improve the performance of the slab memory allocator which is used to allocate memory during logical decoding



© Fujitsu 2023

Compatibility

- Supports a minimum version of Windows 10 for Windows installations
- Removes the promote_trigger_file option to enable the promotion of a standby
  - Users should use the pg_ctl promote command or pg_promote() function to promote a standby
- Remove the server variable vacuum_defer_cleanup_age
  - This has been unnecessary since hot_standby_feedback and replication slots were added.
- Remove libpq support for SCM credential authentication
- Introduced the Meson build system, which will ultimately replace Autoconf

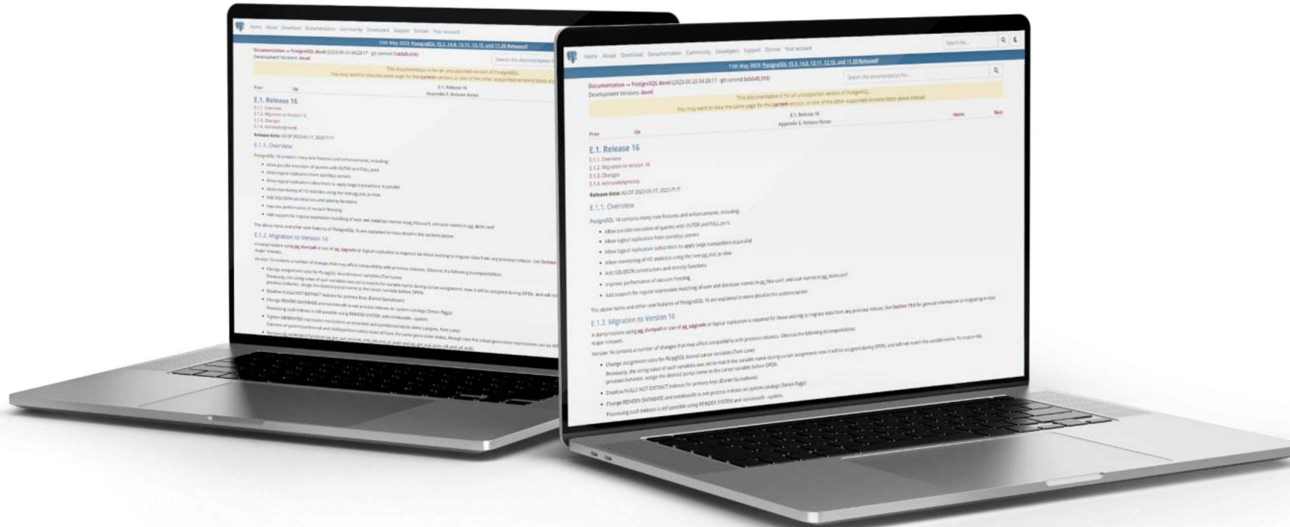


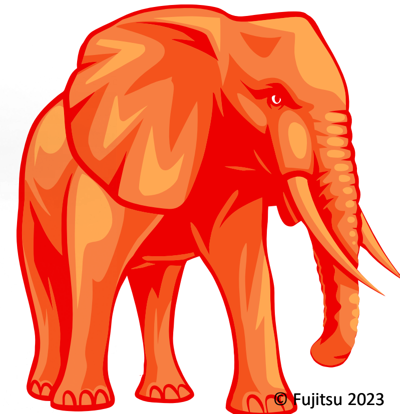
© Fujitsu 2023

Changes in PostgreSQL 16

FUJITSU

- The full list of new/enhanced features and other changes can be found here





© Fujitsu 2023

Agenda

FUJITSU

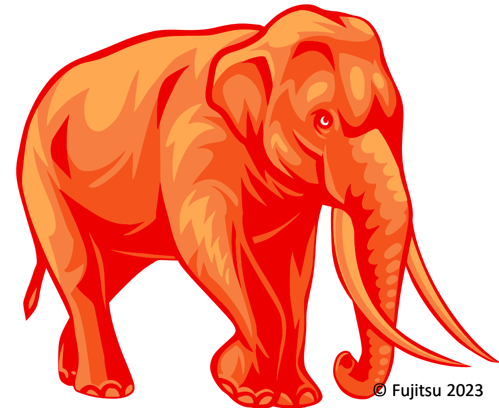
- Key features and performance improvements in PostgreSQL 16
- **PostgreSQL 17 and beyond**

Disclaimer: This section is based on what I could see being proposed in community at this stage

PostgreSQL 17 and beyond

FUJITSU

- Various improvements in Logical Replication
  - DDL Replication
  - Replication of sequences
  - Synchronization of replication slots to allow failover
  - Upgrade of logical replication nodes
  - Reuse of tablesync workers
  - Time-delayed logical replication
  - …
- Reduced number of commands that need superuser privilege
- SQL/JSON improvements to make it more standard compliant

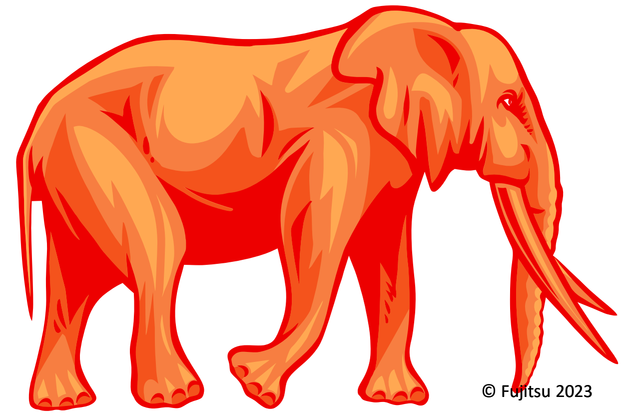


© Fujitsu 2023

PostgreSQL 17 and beyond

FUJITSU

- Transparent column encryption
  - Automatic, transparent encryption and decryption of particular columns in the client
- Asynchronous I/O
  - Will allow prefetching data and will improve system performance
- Large relation files to reduce open/close for huge numbers of file descriptors
- Enhance Table AM APIs
- Amcheck for Gist and Gin indexes
- Improve locking for better scalability



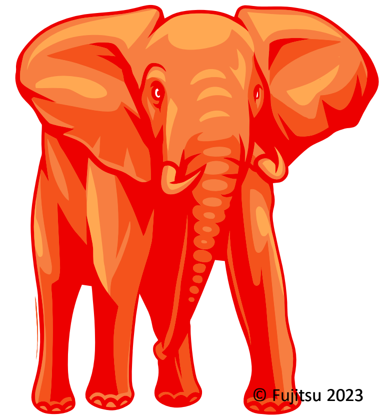
© Fujitsu 2023

PostgreSQL 17 and beyond

3/4

FUJITSU

- Improvements in vacuum technology by using performance data structure
- Improvements in partitioning technology
- Improve statistics/monitoring
- TDE
  - Can help in meeting security compliance in many organizations
- 64bit XIDs
  - Can avoid freezing and reduce the need of autovacuum
- Parallelism
  - Allow parallel-safe initplans
  - Parallelize correlated subqueries

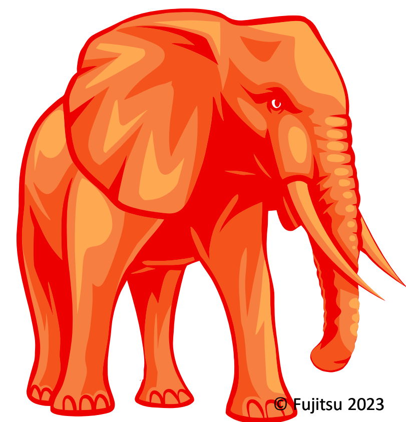


© Fujitsu 2023

PostgreSQL 17 and beyond

FUJITSU

- WAL Size reduction
  - Smaller headers in WAL
- Move SLRU into main buffer pool
- TOAST improvements
  - Custom formats
  - Compression dictionaries
- CI and build system improvements



© Fujitsu 2023

Thank you

PostgreSQL 16 and beyond

Amit Kapila

PostgreSQL Committer and Major Contributor