PostgreSQL 15 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

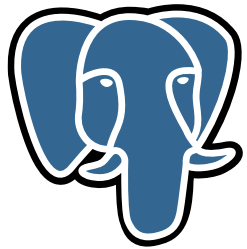
12 — 2019
- Partitioning performance enhancements
- Table Access Methods

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 xacts
- 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)



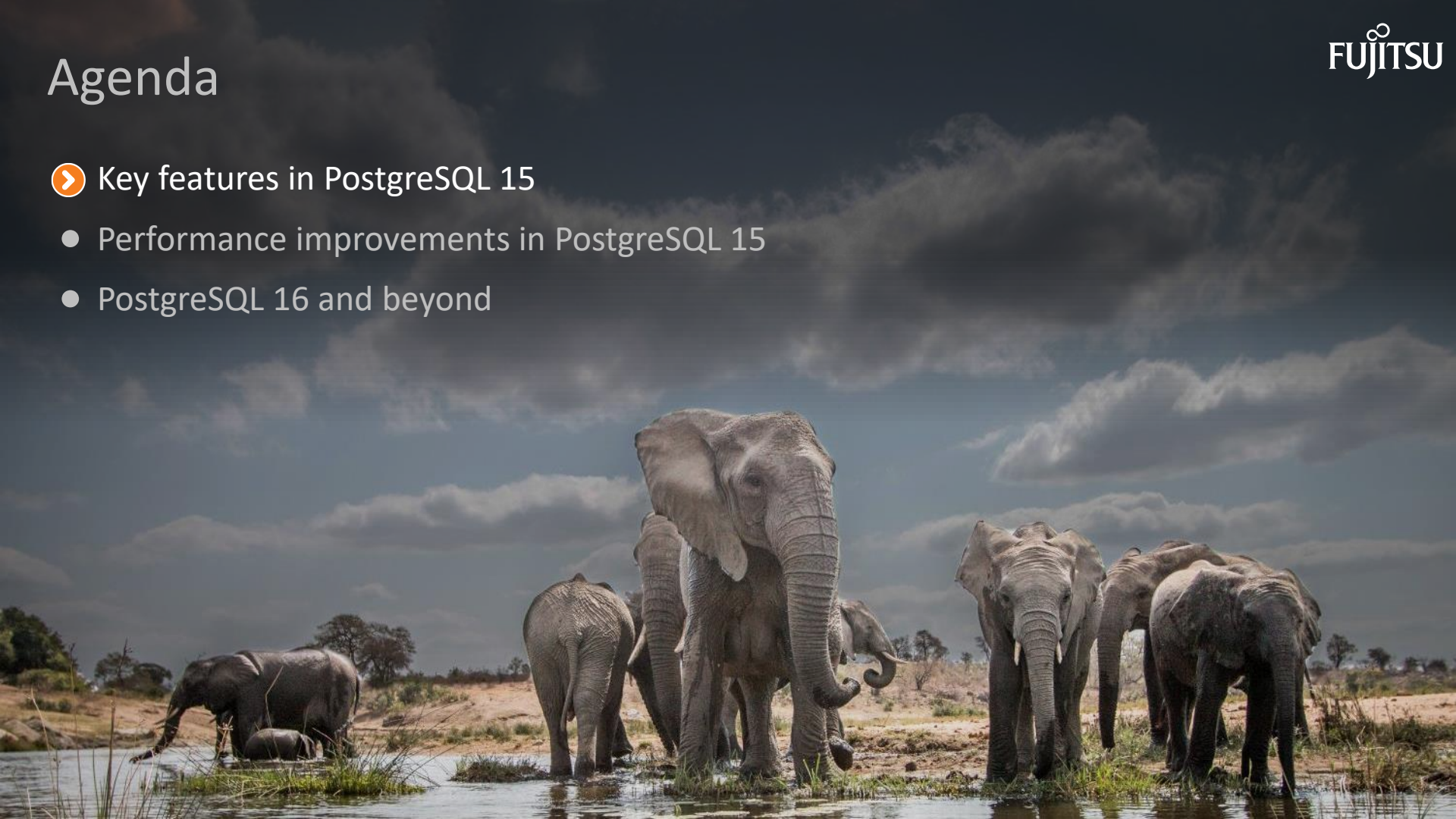
Agenda

FUJITSU

- Key features in PostgreSQL 15
- Performance improvements in PostgreSQL 15
- PostgreSQL 16 and beyond

Agenda

- Key features in PostgreSQL 15
- Performance improvements in PostgreSQL 15
- PostgreSQL 16 and beyond



SQL-Merge

1/2

FUJITSU

- Command handles inserts, updates, and deletes, all in a single transaction
- Attempts to insert a new product – if the given products already exists, then update those, otherwise delete them

```
MERGE INTO TargetProducts Target
USING SourceProducts Source
ON Source.ProductID = Target.ProductID
WHEN NOT MATCHED AND Source.ProductId IS NOT NULL THEN
  INSERT VALUES (Source.ProductID, Source.ProductName, Source.Price)
WHEN MATCHED AND Target.ProductName IN ('Table', 'Desk') THEN
  UPDATE SET ProductName = Source.ProductName, Price = Source.Price
WHEN MATCHED THEN
  DELETE;
```

SQL-Merge

FUJITSU

- Ensures that the join produces at most one candidate change row for each target row
  - Otherwise, it will lead to error "`MERGE command cannot affect row a second time`"
- For each candidate change row, the first clause to evaluate as true is executed
- No more than one WHEN clause is executed for any candidate change row
- One common use case is while trying to maintain Slowly Changing Dimensions (SCD) in a data warehouse. In such cases, one needs to:
  - Insert new records into the data warehouse,
  - Remove records from the warehouse which are not in the source anymore, and
  - Update values in the warehouse which have been updated in the source

Base backups

FUJITSU

- Allows to specify targets for backups
  - Backup location can be specified using `-t target` or `--target=target`
    - `client`: Default value
    - `server`: Stores backup on server
    - `blackhole`: Discards the contents, used only for testing and debugging purpose
  - This option cannot be used with default WAL streaming option `–Xstream`
- Backups can be compressed
  - Server-side compression
  - Client-side compression
  - Compression options: gzip, LZ4, and Zstandard
  - Client-side: gzip was supported prior to 15
  - Allows faster and smaller backups

WAL

FUJITSU

- Full Page Writes can be compressed using LZ4 and Zstandard compression
  - Previously, PGLZ was used
  - User can specify the compression method via GUC `wal_compression`
- WAL archiving via loadable modules
  - Currently, the continuous archiving is done via shell commands
  - This new feature allows users to create custom modules for continuous archiving
  - Although archiving via a shell command is much simpler, a custom archive module will often be considerably more robust and performant
  - For example, if the archiving destination path already exists and has contents identical to source path, we can allow archiving to succeed via custom module
  - Users can use GUC `archive_library` to specify a library that can be used to archive a logfile segment

Stats/Log 1/2

FUJITSU

- Stores statistics in shared memory
  - No more UDP transfer for stats
  - No more writing to temp files
  - The stats in shared memory will be written once at server shutdown
  - The prior stats state is restored after shutdown and restart of server
  - The stats are discarded on restarting server after a crash
- Allows log output in JSON format
  - `log_destination=jsonlog`
  - This is the 3rd type of destination of this kind, after `stderr` and `csvlog`
  - The format is convenient to feed logs to other applications

Stats/Log 2/2

FUJITSU

- New stats in pg_stat_statements
  - I/O timing for temp files
  - JIT counters
- Extended statistics (`CREATE STATISTICS …`) to record statistics for a parent with all its children
  - Regular statistics already tracked this information
  - Improve queries that involve processing the inheritance tree as a whole

Logical replication improvements

FUJITSU

- Adds support for prepared transactions to built-in logical replication

```
CREATE PUBLICATION  mypub FOR ALL TABLES;
CREATE SUBSCRIPTION mysub CONNECTION 'dbname=postgres'
       PUBLICATION  mypub WITH (two_phase = true);
```

  - Reduces the lag to replicate data
  - This provides the base to build conflict-free logical replication
- Allows publication of all tables in a schema

```
CREATE PUBLICATION mypub FOR TABLES IN SCHEMA mysch;
CREATE PUBLICATION mypub FOR TABLE mytab, TABLES IN SCHEMA mysch;
```

  - Tables added later to the listed schemas will also be replicated

Logical replication improvements

FUJITSU

- Allows publication content to be filtered using a WHERE clause

```
CREATE PUBLICATION mypub FOR TABLE mytab WHERE (c1 > 10);
```

  - This can help distribute data among nodes and improve performance by sending data selectively
  - The WHERE clause allows only simple expressions
    - It cannot contain user-defined functions, operators, types, collations, system column references or non-immutable built-in functions
  - If a publication publishes UPDATE or DELETE operations, the row filter WHERE clause must contain only columns that are covered by the replica identity
  - If a publication publishes only INSERT operations, the row filter WHERE clause can use any column

Logical replication improvements

FUJITSU

- Allows publications to publish specific columns for tables

```
CREATE PUBLICATION mypub FOR TABLE mytab (c1, c2);
```

  - The choice of columns can be based on behavioral or performance reasons
  - A column list can contain only simple column references
  - A column list can't be specified if the publication also publishes FOR TABLES IN SCHEMA
  - If a publication publishes UPDATE or DELETE operations, any column list must include the table's replica identity columns
  - If a publication publishes only INSERT operations, then the column list may omit replica identity columns
- Allows logical replication to run as the owner of the subscription
  - Only superusers, roles with bypassrls, and table owners can replicate into tables with row-level security policies

Logical replication improvements

FUJITSU

- Conflict resolution
  - Current methods:
    - By manually removing the conflicting data
    - By skipping the transaction via pg_replication_origin_advance
  - A new method:
    - By specifying LSN of the conflicting transaction

```
ALTER SUBSCRIPTION mysub SKIP (lsn = 0/14C0378)
```

    - Information related to LSN of failed transaction will be available in server logs
- Users can set a parameter to automatically disable replication on conflict
  - Useful for scenarios where a retry could not possibly succeed without human intervention
- New system view – pg_stat_subscription_stats
  - Shows stats about errors which occurred during the application of logical replication changes or during initial table synchronization

FUJITSU

- Allows ICU collations to be set as the default for clusters and databases

```
initdb --locale-provider=icu --icu-locale=en
CREATE DATABASE dbicu LOCALE_PROVIDER icu ICU_LOCALE 'en-u-kf-upper'
```

- Allows unique constraints and indexes to treat NULL values as not distinct

```
CREATE TABLE mytab(c1 text UNIQUE NULLS NOT DISTINCT);
INSERT INTO mytab VALUES('amit');
INSERT INTO mytab VALUES(NULL);
INSERT INTO mytab VALUES(NULL);

ERROR:  duplicate key value violates unique constraint "mytab_c1_key"
    DETAIL:  Key (c1)=(null) already exists.
```

FUJITSU

- ON DELETE, partially SET NULL

```
CREATE TABLE fktable (… FOREIGN KEY (tid, id) REFERENCES pktable ON DELETE SET NULL (id));

postgres=# delete from pktable where id = 2;
DELETE 1
postgres=# select * from fktable;
 tid | id | foo
-----+----+-----
   1 |  1 |   1
   1 |    |   1
(2 rows)
```

- This is useful for multitenant or sharded schemas, where the tenant or shard ID is included in the primary key of all tables but shouldn't be set to null.

- `CREATE DATABASE` new option – `STRATEGY`
  - `WAL_LOG`
    - Database will be copied block by block, and each block will be separately written to the WAL
    - Checkpoints are not required
    - Avoids the impact on overall system due to checkpoint
    - Efficient strategy for small databases
    - Default
  - `FILE_COPY`
    - Current method
    - Writes a small record to the write-ahead log for each tablespace used by the target database
    - Each such record represents copying an entire directory to a new location at the filesystem level
    - Reduces WAL volume for the large template database
    - Performs checkpoint both before and after operation, which can impact system performance

Privileges

FUJITSU

- Granting `SET` and `ALTER SYSTEM` privileges for superuser GUC parameters
  - This allows to set superuser server parameters via non-superuser roles

```
GRANT SET ON PARAMETER wal_compression TO bob;
GRANT ALTER SYSTEM ON PARAMETER wal_compression TO bob;
```

- Predefined role pg_checkpoint
  - Allows members to run CHECKPOINT
  - Previously, checkpoints could only be run by superusers
- Security invoker views
  - Checks permissions for base relations using the privileges of the user of the view
  - Default is to check using the privileges of the view owner

```
CREATE VIEW myview WITH (security_invoker=true) AS SELECT * FROM mytab;
```

Agenda

- Key features in PostgreSQL 15
- Performance improvements in PostgreSQL 15
- PostgreSQL 16 and beyond

FUJITSU

Sorting 1/2

FUJITSU

- Improved performance and reduced memory consumption of in-memory sorts
  - Improved performance of single column sorts by more than 25%
    - Only when the result also contains single column
    - Will be used for `SELECT col1 from mytab ORDER BY col1;`
    - Will not be used for `SELECT col1, col2 from mytab ORDER BY col1;`
  - Reduced memory consumption by using generation memory context
    - We were using memory allocation scheme that rounded requests to the next power of 2
    - Performance improvement depends on tuple size, but up to ~40% improvement is observed
  - Reduced function call overhead by adding specialized sort routines for common datatypes
    - Performance improvement: ~5%

Sorting 2/2

FUJITSU

- Improved performance for sorts that exceed work_mem
  - Switched to a batch sorting algorithm that uses more output streams than before
  - Performance improvement depends on work_mem – the smaller the work_mem, the greater is the improvement
  - Performance improvement: ~40%
- For further details, see blog

postgres_fdw

FUJITSU

- Allows parallel commit on postgres_fdw servers
  - Enabled by the `parallel_commit` option via `CREATE/ALTER SERVER`
  - Commits the transaction in parallel on all foreign servers involved in local transaction
  - This can improve performance of distributed PostgreSQL clusters using postgres_fdw

Partitioning

FUJITSU

- Ordered scans of partitions in more cases
  - Allows optimization when `DEFAULT` and `LIST` partitions containing multiple values gets pruned
  - Uses Append instead of MergeAppend
  - Improved performance by avoiding the need to sort
- Improved planning time for statements where only few of the partitions are relevant

Parallelism

FUJITSU

- Parallelized `SELECT DISTINCT`
  - Introduces two-phase DISTINCT
  - Phase 1 is performed on parallel workers
    - Rows are made distinct there either by hashing or by sort/unique
  - Phase 2 is performed by leader backend
    - Removes duplicate rows that appear due to combining rows for each of the parallel workers

```
SELECT DISTINCT four FROM tenk1;
                    QUERY PLAN
---------------------------------------------------
 Unique
   ->  Sort
         Sort Key: four
         ->  Gather
               Workers Planned: 2
               ->  HashAggregate
                     Group Key: four
                     ->  Parallel Seq Scan on tenk1
```

Recovery/Replay

FUJITSU

- Speeded up recovery/replay by prefetching needed file contents
  - `recovery_prefetch`: When enabled, looks ahead in the WAL and try to initiate asynchronous reading of referenced data blocks not yet cached in our buffer pool
  - Works where `posix_fadvise()` is available
  - `wal_decode_buffer_size`: Maximum distance to read ahead in the WAL to prefetch referenced data blocks

Vacuum

FUJITSU

- Allows vacuum to be more aggressive in setting the oldest `frozenxid`
  - Before 15, vacuum set it to whatever value was used to determine which tuples to freeze – the FreezeLimit cutoff
  - Now, we set it to value <= the oldest extant XID remaining in the table
  - This can be much more recent than FreezeLimit
  - This will help in reducing the times anti-wraparound vacuum is invoked for certain workloads

Changes in PostgreSQL 15

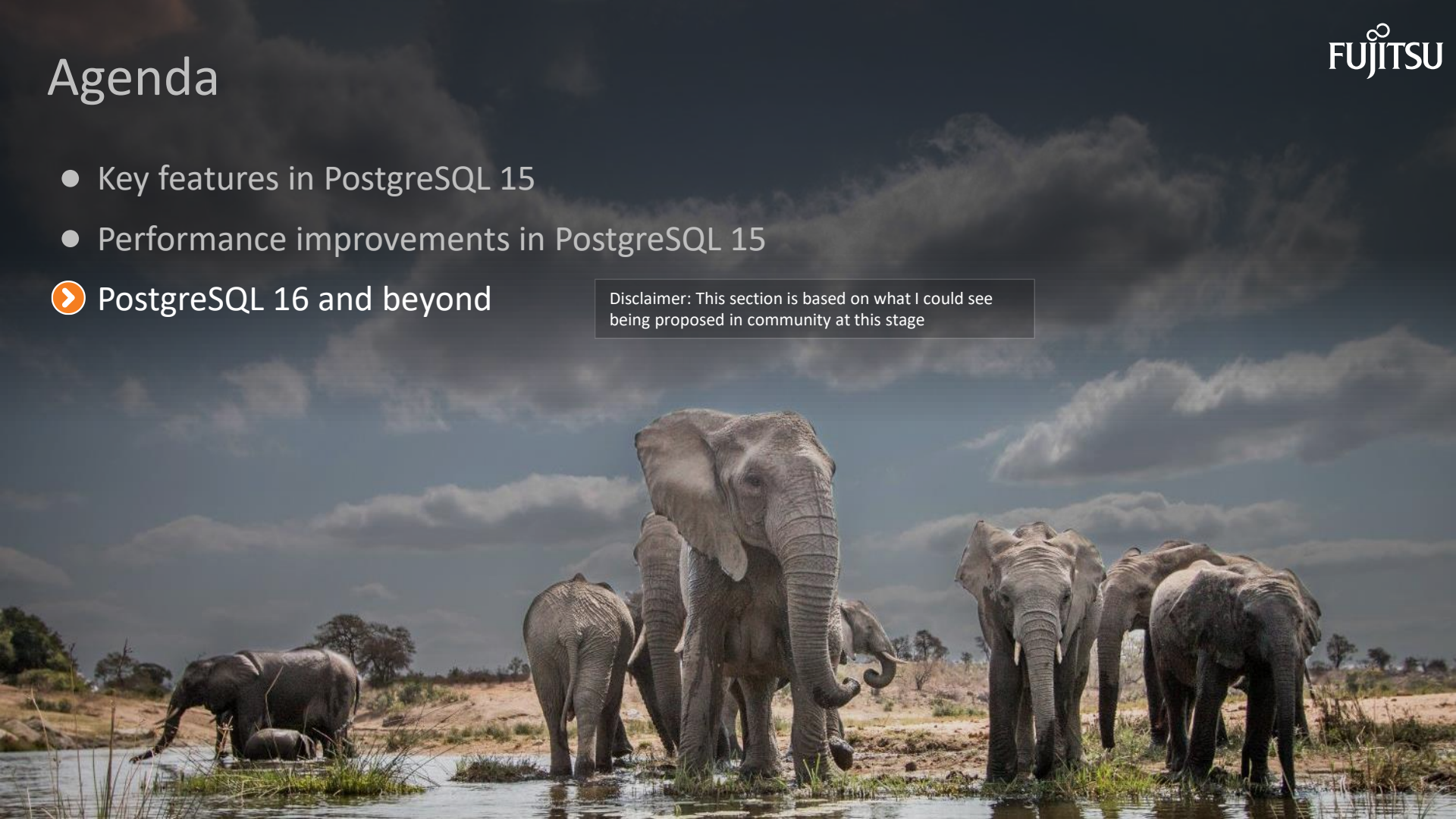
FUJITSU

- New features and enhancements
  - Support for the SQL MERGE command
  - Selective publication of tables' contents within logical replication publications, through the ability to specify column lists and row filter conditions
  - More options for compression, including support for Zstandard (zstd) compression
    - Includes support for performing compression on server side during pg_basebackup
  - Support for structured server log output using the JSON format
  - Performance improvements, particularly for in-memory and on-disk sorting
- The full list of new/enhanced features and other changes can be found here

Agenda

- Key features in PostgreSQL 15
- Performance improvements in PostgreSQL 15
- PostgreSQL 16 and beyond

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



PostgreSQL 16 and beyond

FUJITSU

- Various improvements in Logical Replication
  - Allow same table replication by filtering based on origins
  - Parallel Apply
  - Replication of sequences
  - Enable logical replication from standby
  - DDL Replication
  - Use of indexes on subscriber when publisher has specified replica identity full
  - Replication of other objects like LOBs
  - …
- Reduced number of commands that need superuser privilege
- SQL/JSON improvements to make it more standard compliant

PostgreSQL 16 and beyond

FUJITSU

- Transparent column encryption
  - Automatic, transparent encryption and decryption of particular columns in the client
- Change build infrastructure by replacing it with Meson build system
  - Developer-oriented feature
- Asynchronous I/O
  - Will allow prefetching data and will improve system performance
- Direct I/O
  - Will bypass the OS cache and lead to better performance in some cases
- Various improvements in Hash indexes
  - Allow unique indexes
  - Allow multi-column indexes

PostgreSQL 16 and beyond

FUJITSU

- Improvements in vacuum technology by using performance data structure, advancing relfrozenxid earlier, and by reducing WAL volume
- Improvements in partitioning technology
- Improve statistics/monitoring
- 64bit XIDs
  - Can avoid freezing and reduce the need of autovacuum
- TDE
  - Can help in meeting security compliance in many organizations
- Incremental maintenance of materialized views

Thank you

PostgreSQL 15 and beyond

Amit Kapila

PostgreSQL Committer and Major Contributor