A look at the Elephants Trunk

PostgreSQL 15

PGConf.NYC 2022
New York, USA

Magnus Hagander
magnus@hagander.net
Magnus Hagander

- Redpill Linpro
  - Principal database consultant
- PostgreSQL
  - Core Team member
  - Committer
  - PostgreSQL Europe
PostgreSQL 15
Development schedule

- June 2021 - branch 14
- July 2021 - CF1
- September 2021 - CF2
- November 2021 - CF3
- January 2022 - CF4
- March 2022 - CF5
- June 2022 - Beta2
New features

- DBA and administration
- SQL and developer
- Backup and replication
- Performance
Breaking changes
Old version support

- Support for pre-9.2 removed
  - psql
  - pg_dump/pg_dumpall
- (but you had updated, right?)
Python 2

- Python 2 support in pl/python dropped
- (Python 2 EOL in January 2020!)
public schema

- *public* no longer has create permissions!
  - Only *pg_database_owner*
- *public* retains *usage* permissions
Exclusive backup mode

- Removed
- Deprecated for a long time
  - Unsafe!
- Use non-exclusive mode!
  - Or pg_basebackup
New features

- DBA and administration
- SQL and developer
- Backup and replication
- Performance
SSL

- Allow root-owned SSL private keys in libpq
  - Already allowed in backend
Predefined roles

- pg_checkpoint
  - Allowed to run CHECKPOINT
Permissions on GUCs

- Reduce permissions on superuser gucs
Permissions on GUCs

- Reduce permissions on superuser gucs

```
GRANT SET
ON PARAMETER track_functions TO joe
```
Permissions on GUCs

- Reduce permissions on superuser gucs

GRANT SET
ON PARAMETER track_functions TO joe

GRANT ALTER SYSTEM
ON PARAMETER track_functions TO joe
Memory sizing

- New GUC: `shared_memory_size`
- New GUC: `shared_memory_size_in_huge_pages`
Memory sizing

- New GUC: `shared_memory_size`
- New GUC: `shared_memory_size_in_huge_pages`

```
$ postgres -C shared_memory_size
143
$ postgres -C shared_memory_size_in_huge_pages
72
```
pg_stat_statements

- I/O timing for temp files
- JIT counters
New wait events

- ArchiveCommand
- ArchiveCleanupCommand
- RestoreCommand
- RecoveryEndCommand
Logging changes

- Startup process logs what it's doing
  - Every `log_startup_progress_interval` (10s)
- New defaults:
  - `log_autovacuum_min_duration = 10 min`
  - `log_checkpoints = on`
JSON logging

- `log_destination = jsonlog`
- Like csvlog
  - But json
  - *Always written to file*
Security invoker views

- Checks permissions with callers privileges
- Default: check with view creators

```sql
CREATE VIEW myview
WITH (security_invoker=true)
AS SELECT * FROM sometable WHERE x=3
```
ICU locales

- Global locale provider
  - Per cluster
  - Per database

```
initdb --locale-provider=icu --icu-locale=sv_SE

CREATE DATABASE foo TEMPLATE template0
  LOCALE_PROVIDER 'icu'
  ICU_LOCALE 'fi'
```
New features

- DBA and administration
- SQL and developer
- Backup and replication
- Performance
Numeric

- Negative scale

```
SELECT 1234::numeric(5,1),
       1234::numeric(5,0),
       1234::numeric(5,-1);
1234.0 |   1234 |   1230
```

- Scale greater than precision

```
select 0.01::numeric(2,3);
0.010
```
ON DELETE

- Partial set NULL

CREATE TABLE xyz (
  ..., 
  FOREIGN KEY (col1, col2, col3)
    REFERENCES othertable
  ON DELETE SET NULL (col2, col3)
)
UNIQUE vs NULL

1. `CREATE TABLE u (a int UNIQUE);`
2. `INSERT INTO u VALUES (1);`
3. `INSERT INTO u VALUES (NULL);`
4. `INSERT INTO u VALUES (NULL);`
UNIQUE vs NULL

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CREATE TABLE u (a int UNIQUE);</td>
</tr>
<tr>
<td>2</td>
<td>INSERT INTO u VALUES (1);</td>
</tr>
<tr>
<td>3</td>
<td>INSERT INTO u VALUES (NULL);</td>
</tr>
<tr>
<td>4</td>
<td>INSERT INTO u VALUES (NULL);</td>
</tr>
<tr>
<td></td>
<td>Code</td>
</tr>
<tr>
<td>---</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>1</td>
<td><code>CREATE TABLE u (a int UNIQUE);</code></td>
</tr>
<tr>
<td>2</td>
<td><code>INSERT INTO u VALUES (1);</code></td>
</tr>
<tr>
<td>3</td>
<td><code>INSERT INTO u VALUES (NULL);</code></td>
</tr>
<tr>
<td>4</td>
<td><code>INSERT INTO u VALUES (NULL);</code></td>
</tr>
</tbody>
</table>
UNIQUE vs NULL

1 CREATE TABLE u (a int UNIQUE);
2 INSERT INTO u VALUES (1);
3 INSERT INTO u VALUES (NULL);
4 INSERT INTO u VALUES (NULL);
<table>
<thead>
<tr>
<th></th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>CREATE TABLE u (a int UNIQUE);</code></td>
</tr>
<tr>
<td>2</td>
<td><code>INSERT INTO u VALUES (1);</code></td>
</tr>
<tr>
<td>3</td>
<td><code>INSERT INTO u VALUES (NULL);</code></td>
</tr>
<tr>
<td>4</td>
<td><code>INSERT INTO u VALUES (NULL);</code></td>
</tr>
</tbody>
</table>

- What happens?
UNIQUE vs NULL

1. CREATE TABLE u (a int UNIQUE NULLS NOT DISTINCT);
2. INSERT INTO u VALUES (1);
3. INSERT INTO u VALUES (NULL);
4. INSERT INTO u VALUES (NULL);
UNIQUE vs NULL

1. CREATE TABLE u (a int UNIQUE NULLS NOT DISTINCT);
2. INSERT INTO u VALUES (1);
3. INSERT INTO u VALUES (NULL);
4. INSERT INTO u VALUES (NULL);
### UNIQUE vs NULL

<table>
<thead>
<tr>
<th></th>
<th>SQL Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>CREATE TABLE u (a int UNIQUE NULLS NOT DISTINCT);</code></td>
</tr>
<tr>
<td>2</td>
<td><code>INSERT INTO u VALUES (1);</code></td>
</tr>
<tr>
<td>3</td>
<td><code>INSERT INTO u VALUES (NULL);</code></td>
</tr>
<tr>
<td>4</td>
<td><code>INSERT INTO u VALUES (NULL);</code></td>
</tr>
</tbody>
</table>
CREATE TABLE u (a int UNIQUE NULLS NOT DISTINCT);
INSERT INTO u VALUES (1);
INSERT INTO u VALUES (NULL);
INSERT INTO u VALUES (NULL);
CREATE TABLE u (a int UNIQUE NULLS NOT DISTINCT);

1  INSERT INTO u VALUES (1);
2  INSERT INTO u VALUES (NULL);
3  INSERT INTO u VALUES (NULL);
4  INSERT INTO u VALUES (NULL);

ERROR:  duplicate key value violates unique constraint "u_a_ke
DETAIL:  Key (a)=(null) already exists.
MERGE
MERGE

• *Not* a replacement for **ON CONFLICT**
• Different problem, different solution
• **ON CONFLICT** is for *upsert*
• Merge is for, well, merging
  ■ *Not* atomic!
MERGE

- JOINs a target with a source
- Defines rules for how to transform data
- Modifies or adds to target
MERGE INTO target t

USING changes c

ON t.typeid = c.typeid

WHEN NOT MATCHED AND c.delta > 0 THEN

INSERT VALUES (c.name, c.delta)

WHEN MATCHED AND t.num + c.delta > 0 THEN

UPDATE SET num = t.num + c.delta

WHEN MATCHED THEN

DELETE
MERGE INTO target t
USING changes c
ON t.typeid = c.typeid
WHEN NOT MATCHED AND c.delta > 0 THEN
  INSERT VALUES (c.name, c.delta)
WHEN MATCHED AND t.num + c.delta > 0 THEN
  UPDATE SET num = t.num + c.delta
WHEN MATCHED THEN
DELETE
MERGE INTO target t
USING changes c
ON t.typeid = c.typeid
WHEN NOT MATCHED AND c.delta > 0 THEN
  INSERT VALUES (c.name, c.delta)
WHEN MATCHED AND t.num + c.delta > 0 THEN
  UPDATE SET num = t.num + c.delta
WHEN MATCHED THEN
DELETE
MERGE INTO target t
USING changes c
ON t.typeid = c.typeid
WHEN NOT MATCHED AND c.delta > 0 THEN
  INSERT VALUES (c.name, c.delta)
WHEN MATCHED AND t.num + c.delta > 0 THEN
  UPDATE SET num = t.num + c.delta
WHEN MATCHED THEN
DELETE
MERGE INTO target t
USING changes c
ON t.typeid = c.typeid
WHEN NOT MATCHED AND c.delta > 0 THEN
  INSERT VALUES (c.name, c.delta)
WHEN MATCHED AND t.num + c.delta > 0 THEN
  UPDATE SET num = t.num + c.delta
WHEN MATCHED THEN
DELETE
MERGE

1  MERGE INTO target t
2  USING changes c
3  ON t.typeid = c.typeid
4  WHEN NOT MATCHED AND c.delta > 0 THEN
5      INSERT VALUES (c.name, c.delta)
6  WHEN MATCHED AND t.num + c.delta > 0 THEN
7      UPDATE SET num = t.num + c.delta
8  WHEN MATCHED THEN
9      DELETE
New features

- DBA and administration
- SQL and developer
- Backup and replication
- Performance
Logical Replication

- Two-phase commit
  - Send prepared transactions
Logical Replication

• Publish all tables in schema

CREATE PUBLICATION pub2
FOR ALL TABLES IN SCHEMA myschema
Logical Replication

- Row filtering

CREATE PUBLICATION pub1
FOR TABLE xyz
WHERE (col1 > 10)
Logical Replication

• Column filtering

CREATE PUBLICATION pub2
FOR TABLE xyz(col1, col2, col3)
Logical Replication

- More statistics
  - `pg_stat_subscription_stats`
- `disable_on_error` option
Logical Replication

- Skip transaction
  - On failure (well..)
  - Skip change, continue replication

ALTER SUBSCRIPTION mysub
SKIP (lsn = 0/150B868)
Base backups

- Server side compression
- Compress-then-send
Base backups

- Server side compression
- Compress-then-send
- Client-side decompression
  - Replicas over metered connections
Base backups

- Base backup targets
  - client
  - server
  - (blackhole)
Log archiving

- Loadable modules
  - `archive_library = 'xyz'`
  - Can be made more efficient
    - Shell commands have large overhead
  - Easier to make reliable
New features

- DBA and administration
- SQL and developer
- Backup and replication
- Performance
LZ4 + zstd

- wal_compression
- Base backups
  - Client-side
  - Server-side
- pg_receivewal
  - LZ4 only
Parallel query

- Parallel DISTINCT
Partitioning

- Ordered partition scans
  - More cases
  - LIST partitions
Monotonic window functions

- Smarter planner!
Monotonic window functions

- Smarter planner!

```sql
SELECT * FROM (
    SELECT g,
        row_number() over(order by g) AS rn
    FROM x
) t
WHERE rn < 5;
```
Statistics

- Stored in shared memory
- No longer temp files!
- No longer UDP to transfer!
- No more stats collector!
Recovery prefetch

- Initiate async I/O for future WAL
- (with fadvise)
- OS dependent
There's always more
There's always more

- Lots of smaller fixes
- Performance improvements
- etc, etc
- Can't mention them all!
Please help!

- Beta version available!
- Download and test!
  - apt packages available
  - rpm/yum packages available
Thank you!

Magnus Hagander
magnus@hagander.net
@magnushagander
https://www.hagander.net/talks/