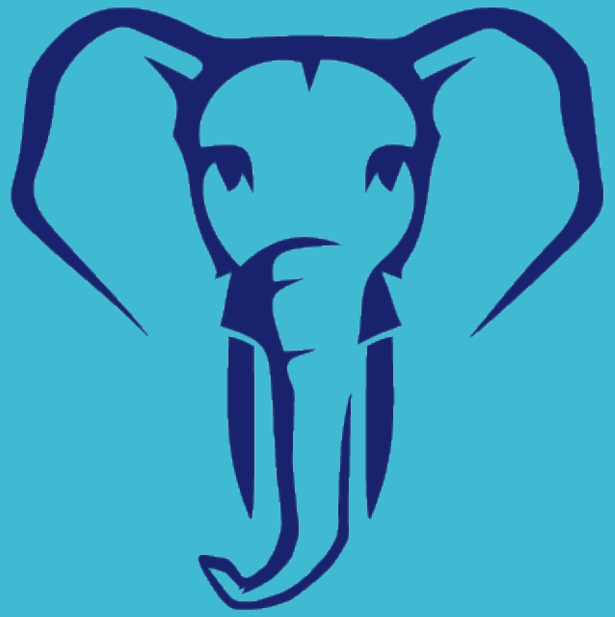
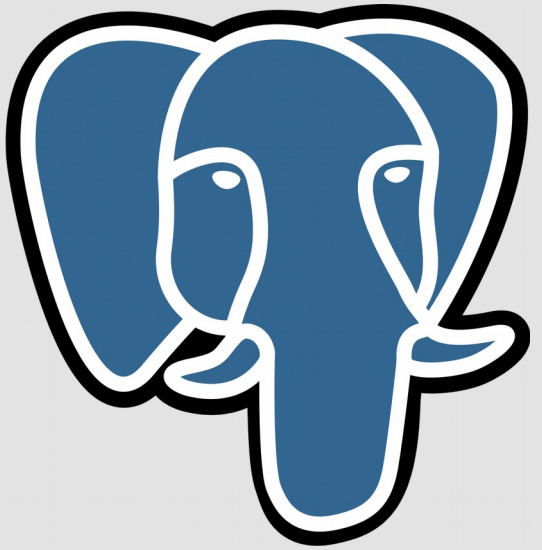
PostgreSQL V12



People

Postgres

Data



Participate

Your Speaker

- Joshua D. Drake (JD)
- Founder United States PostgreSQL
- Founder Command Prompt, Inc.
  - https://postgresconf.org/
- Postgres Conference Co-Chair (Vision and Purpose)
  - https://postgresconf.org/
- in/postgres

Your Next Conference

V12

We will be discussing V12 major Features as well as Migration considerations!

Partitioning

- V10 brought about partitioning (with Syntax)
- V11 brought:
  - Primary keys
  - Foreign Keys (To foreign tables)
  - Auto Indexing
  - Default partitions
  - Better aggregation planning
- V12 brings:
  - Efficient planning for thousands of partitions
  - Foreign Keys (To a child partition)
  - Improved Copy into Partitions
  - Partitions boundaries now support expressions
  - Functions for investigating partitions

Partitioning Functions

- pg_partition_root()
  - returns the top-most parent of a partition tree
- pg_partition_ancestors()
  - reports all ancestors of a partition
- pg_partition_tree()
  - displays information about the partitions

JSON

- PostgreSQL V12 now support JSON ‘PATH’, the SQL standard JSON language
  - More information:
    - https://www.postgresql.org/docs/12/functions-json.html#FUNCTIONS-SQLJSON-PATH
- Also check out Zson
  - https://github.com/postgrespro/zson

Generated Columns

- GENERATED ALWAYS AS (generation_expr) STORED
  - This clause creates the column as a generated column.
  - The keyword STORED is required to signify that the column will be computed on write and will be stored on disk.
  - The generation expression can refer to other columns in the table, but not other generated columns. Any functions and operators used must be immutable.

```
CREATE TABLE (one int4, two int4 GENERATED ALWAYS AS (one * 5)
STORED);

CREATE TABLE foo (created_date date, expiration_date date
GENERATED ALWAYS AS (created_date + '30 days'::interval)
STORED);
```

Btree

- A reduction in multi-column index size
- Better performance with indexes that contain many duplicates
- Increased efficiency of Vacuum on indexes with many duplicates
- Locking requirements have been reduced on index updates
- INSERTS are now faster with BTREE indexes due to a reduction in locking overhead

Most-Common-Value Statistics

```
CREATE STATISTICS stts3 (mcv) ON state, city FROM zipcodes;

ANALYZE zipcodes;

SELECT m.* FROM pg_statistic_ext,
pg_mcv_list_items(stxmcv) m WHERE stxname = 'stts3';

index  | values      | nulls | frequency | base_frequency
-------+-----------------------------+-------+-----------+
0 | {Washington, DC} | {f,f} | 0.003467  | 2.7e-05
1 | {Apo, AE}        | {f,f} | 0.003067  | 1.9e-05
2 | {Houston, TX}    | {f,f} | 0.002167  | 0.000133
3 | {El Paso, TX}    | {f,f} | 0.002     | 0.000113
4 | {New York, NY}   | {f,f} | 0.001967  | 0.000114
5 | {Atlanta, GA}    | {f,f} | 0.001633  | 3.3e-05
6 | {Sacramento, CA} | {f,f} | 0.001433  | 7.8e-05
7 | {Miami, FL}      | {f,f} | 0.0014    | 6e-05
8 | {Dallas, TX}     | {f,f} | 0.001367  | 8.8e-05
9 | {Chicago, IL}    | {f,f} | 0.001333  | 5.1e-05
```

Inline Many CTE Queries

Many common table expressions (CTE) can now be inlined:

Note: This can cause unexpected results with complicated CTEs that have workarounds to previous performance behaviors.

Prepared Plan Control

In V12, PostgreSQL will optimize planned queries over a series of 6 executions. In short it takes the first 5 executions with a custom plan. On the sixth execution it will check if a generic plan would performed as well or better by comparing the planner costs between the two. If the generic plan will perform as well the prepared statement will use that plan going forward.

There is a new session variable which enables some forced behavior, plan_cache_mode. The parameter can be set in the normal fashion of using “SET”. It has the options of ‘force_custom_plan’ and ‘auto’. The default is ‘auto’ resembles the behavior in versions less than 11.

Note: This is all about PREPARE/BIND/EXECUTE to protect against SQL injection attacks and save on planning time.

JIT:
Just in Time

- Optional Just-in-Time (JIT) compilation for some SQL code, speeding evaluation of expressions is now ‘ON’ by default.
- Useful for data warehouse queries
- Requires LLVM (not GCC)
  - For more information:
    - https://www.postgresql.org/docs/12/jit-reason.html

JIT Uses:

The JIT expression compiler currently works best in the following situation:

- the query contains several complex expression such as aggregates.
- the query reads a fair amount of data but isn’t starved on IO resources.
- the query is complex enough to warrant spending JIT efforts on it.

Example benefits explained here:

- https://www.citusdata.com/blog/2018/09/11/postgresql-11-just-in-time/

Checksum Control

- With version 9.3 (can you believe 2013!?), you could initialize a PostgreSQL cluster with checksums for better data integrity checking.
  - Cluster wide, a one time irreversible operation
  - Huge performance hit
- With version v12 in 2019, you will be able to turn checksums on or off while the database is not in service (off).
- Online enable / disable is planned for a future release

pg_\ checksums

- We now have a new tool to enable or disable checksums:
  - pg_checksums
    - --enable / --disable
    - --progress (display progress of operation)
    - --check
    - --filenode (Only validate the checksums in the relation of a specific filenode)
    - --no-sync (Faster but has potential for silent failure)
- Do not run this on a running cluster
- Can take a long time
- Can be rerun if accidentally killed
- Formally pg_verify_checksums was used

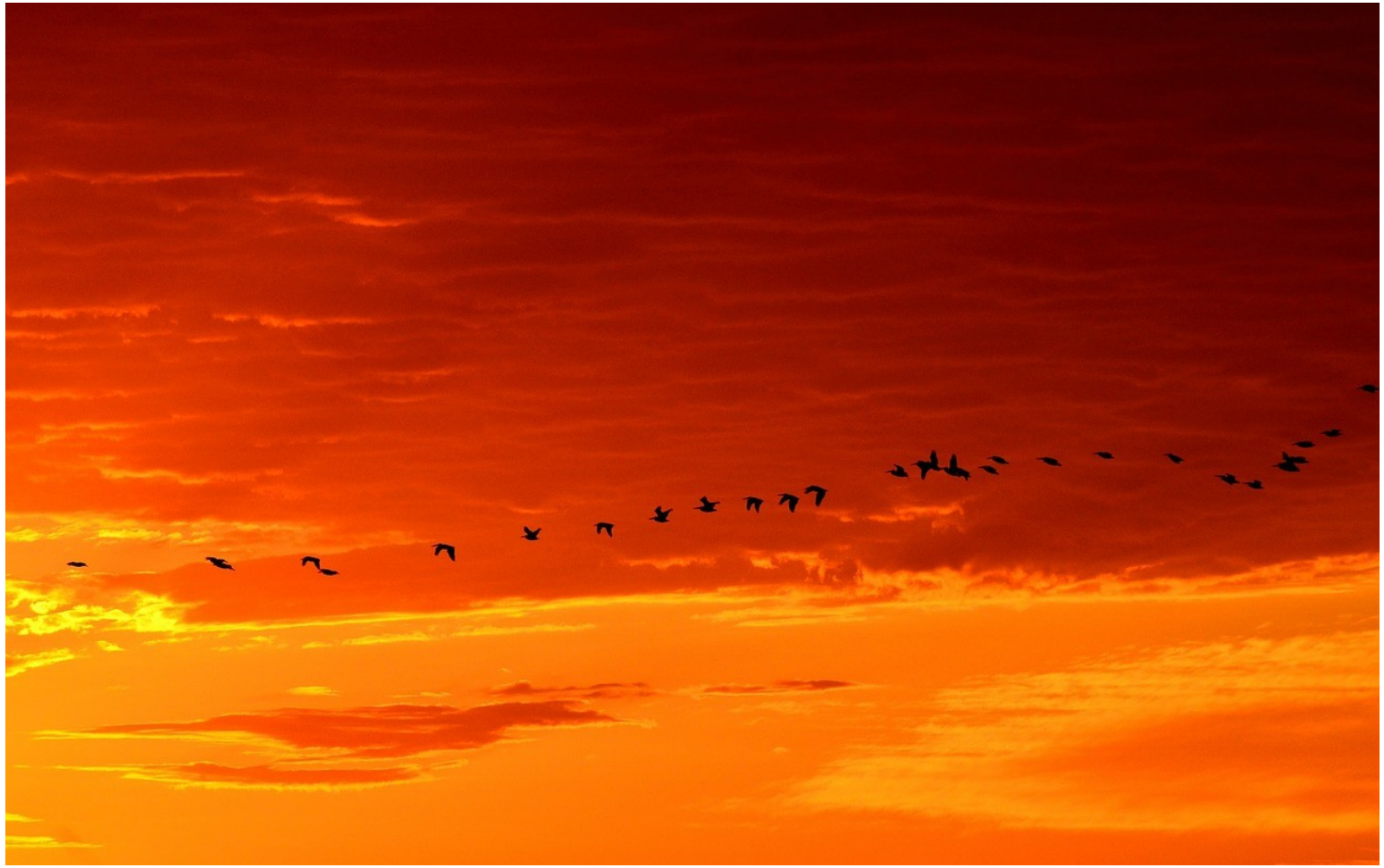
Concurrent REINDEX

- REINDEX CONCURRENTLY
  - Slower than normal REINDEX:
    - Multiple scans of the table
    - Must wait for any queries that might use the index complete
  - The following steps occur when using CONCURRENTLY, each in an isolated transaction
    - A new temporary index definition is added to the catalog pg_index. This definition will be used to replace the old index.
    - A first pass to build the index is done for each new index. Once the index is built, its flag pg_index.indisready is switched to “true” to make it ready for inserts, making it visible to other sessions once the transaction that performed the build is finished.
    - A 2nd pass, adding tuples that were added created during the first pass.
    - All the constraints that refer to the index are changed to refer to the new index definition, and the names of the indexes are changed.
    - The old indexes have pg_index.indisready switched to “false” to prevent any new tuple insertions.
    - The old indexes are dropped.

The kahuna

- Pluggable storage
  - In PostgreSQL world this is called “ACCESS METHODS”
- The ability to create new table and index access methods such as Zheap
  - Provide better control over bloat. zheap will prevent bloat by allowing in-place updates in common cases and by reusing space as soon as a transaction that has performed a delete or non-in-place-update has committed. In short, with this new storage, whenever possible, we’ll avoid creating bloat in the first place.
  - Reduce write amplification both by avoiding rewrites of heap pages and by making it possible to do an update that touches indexed columns without updating every index.
  - Reduce the tuple size by shrinking the tuple header and eliminating most alignment padding.
    - tl;dr; NO VACUUM
- ZedStore
  - in-core columnar storage with similar benefits to Zheap

Migration Considerations



Migrating

- OID
- Data type removals
- Extensions no longer in existence
- Recovery.conf changes
- Geometric function and data type changes
- Changes to REAL / DOUBLE Precision
- pg_restore
- Btree
- Other compatibility issues

OID

- The special behavior of the OID column is not defunct.
  - Previously an OID column would only be automatically created if you specified ‘with OIDS’ when creating a table. This has been removed.
  - Columns can still be explicitly created as type OID.
  - If you rely on an OID column and use SELECT * will result in the OID column being exposed, potentially breaking applications as the OID column would be unexpected in the output.

Datatype removal

- The following data types have been removed as of v12:
  - abstime,
  - reltime
  - tinterval

Deceased Extensions

- The timetravel extension has been removed
  - Was used to see history of a tuple
  - has been removed

recovery\.conf

- With the release of v12, the recovery.conf is now **within** the postgresql.conf.
  - Previous behavior of a separate recovery.conf file is no longer supported
    - if a previous recovery.conf file exists, PostgreSQL will not start
  - files recovery.signal and standby.signal are used to switch into non-primary mode.
    - To start recovery mode create a file called recovery.signal
    - If both standby.signal and recovery.signal exist, the standby mode takes precedence
  - The option “trigger_file” has been renamed to promote_trigger_file
  - The standby_mode option has been removed
  - Do not allow conflicting recovery_target* settings
    - Specifically only one recovery target can be set
  - recovery_target_timeline now defaults to latest
    - The previous value was current

Geometric function and data type changes

- Geometric function have been refactored to more accurate but slightly different results from previous releases
- The behavior of the line data type has been improved
  - As has error reporting for this data type

Real and Double Precision

- Previous, float values would have output rounded to 6 or 15 decimals by default. The new behavior only outputs the number of digits required to preseve the exact binary value.
  - To restore compatible behavior set extra_float_digits to 0
  - New behavior is a performance improvement

pg_restore

- pg_restore now requires -f – in order to output to STDOUT
  - This may break some scripts
  - Is actually, finally, technically correct

BTREE

- BTREE indexes now have a maxium index length that is eight bytes less
  - Could cause REINDEX to fail as values may be too large

Other compatible issues

- The data type ‘name’ can now use non-C collations
- DROP IF EXISTS FUNCTION/PROCEDURE/AGGREGATE/ ROUTINE will now error if no arguments are supplied and there are multiple matching objects
- Removal of pg_constraint.consrc
- Removal of pg_attrdef.adsrc
- You are no longer able to disable dynamic shared memory
  - Specifically dynamic_shared_memory_type is now required and can not be set to none

Other Notables

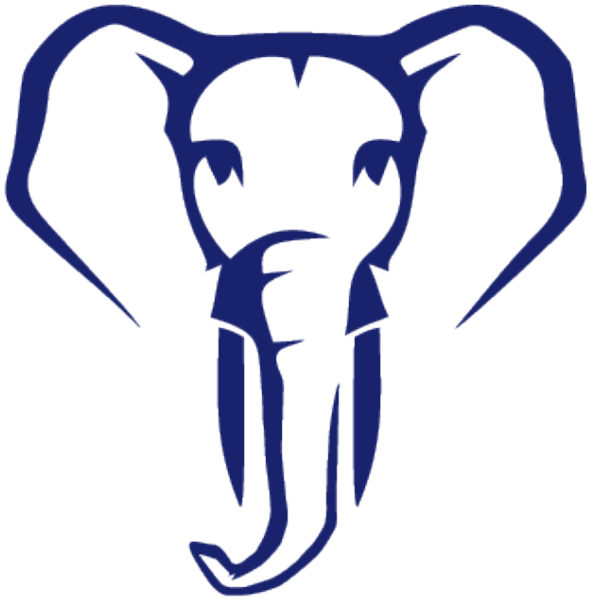
- If you alter the timestamp or timestamptz data type to be one of the other, there is no table rewrite required if the session time is set to ‘UTC’
- Parallel queries are now allowed in ISOLATION LEVEL ‘SERIALIZABLE’
- CREATE INDEX and REINDEX now support progress reporting.
  - See pg_stat_progress_create_index
- CLUSTER AND VACUUM FULL now support progress reporting
  - See pg_stat_progress_cluster
- GSSAPI client and server encryption support
- Server variables can now use fractional input
  - SET work_mem = ‘26.4MB’

Other Notables

- The following replication parameters can be changed with a SIGHUP (reload)
  - archive_cleanup_command
  - promote_trigger_file
  - recovery_end_command
  - recovery_min_apply_delay
- Allow replication slots to be copied
  - pg_copy_physical_replication_slot()
  - pg_copy_logical_replication_slot()
- Allow CHAIN of commit
  - If AND CHAIN is specified on COMMIT, a new transaction is immediately started with the same transaction characteristics

Thanks!

- To:
  - Postgres Conference: For the foundation of Vision and Purpose
  - PostgreSQL.org for the best documentation of any open source project
  - Montreal Postgres for attending
  - To the fantastic People, Postgres, Data community!
  - PostgresWarrior for taking all the “other” battles that allow me to focus on Vision and Purpose!



People

Postgres

Data