25 Interesting features of PostgreSQL 12

PostgreSQL 12

Jobin Augustine
Senior Support Engineer - PostgreSQL
Percona

PostgreSQL Conference Europe, Oct 2019, Milan, Italy

PostgreSQL 12

- Partitioning improvements
- Indexing Improvements
- Standby improvements
- Optimizer Improvements
- Monitoring Improvements
- Security / Authentication improvements
- Server Configuration configuration
- General performance and optimization
- New features

- New functionality
- Client library improvements
- deprecated / obsolete features
- tools - psql, pgbench, vacuumdb, pg_ctl, pg_upgrade, pg_checksums, pg_rewind, pg_dump, pg_dumpall,pg_restore

Internal Infrastructure change for enabling the storage engine is not covered in this talk

Partitioning

- Synthetic benchmark claims upto **76 times** improvements for **SELECT** and **420 times** for **UPDATEs**

```
select count(*) from TRADING WHERE trade_ts between '2019-02-02' and '2019-02-03';
```

Usual User Complaints

- Partition Pruning Problems
- Slower performance than Unpartitioned table
- Performance slow down as the number of partitions increases

```
Planning time: 150.562 ms
Execution time: 5.663 ms
```

Partitioning - Select

PG11:

Planning Time: 49.866 ms

Execution Time: 0.093 ms

PG12:

Planning Time: 0.276 ms

Execution Time: 0.083 ms

Partitioning - not just select

PG11

```
Update on trading  (cost=0.42..8.44 rows=1 width=36) (actual time=0.032..0.032 rows=0 loops=1)
  Update on trading_p1962
  ->  Index Scan using trading_p1962_trade_ts_idx on trading_p1962  (cost=0.42..8.44 rows=1 widt
        Index Cond: (trade_ts = '1962-01-01 00:02:00'::timestamp without time zone)
Planning Time: 148.643 ms
Execution Time: 0.068 ms
```

PG12

```
Update on trading  (cost=0.28..8.30 rows=1 width=36) (actual time=0.191..0.191 rows=0 loops=1)
  Update on trading_p1962
  ->  Index Scan using trading_p1962_trade_ts_idx on trading_p1962  (cost=0.28..8.30 rows=1 width
        Index Cond: (trade_ts = '1962-01-01 00:02:00'::timestamp without time zone)
Planning Time: 0.697 ms
Execution Time: 0.349 ms
```

Partitioning - Inserts

- Less locking
- Consistent performance with large number of partitions

Removing [Merge]Append nodes

which contain a single subpath

PG 11

```
Aggregate  (cost=25.98..25.99 rows=1 width=8) (actual time=0.010..0.011 rows=1 loops=1)
  ->  Append  (cost=16.45..25.97 rows=6 width=0) (actual time=0.007..0.007 rows=0 loops=1)
        ->  Bitmap Heap Scan on trading_p2019_02  (cost=16.45..25.94 rows=6 width=0) (actual
              Recheck Cond: ((trade_ts >= '2019-02-02 00:00:00'::timestamp without time zone)
              ->  Bitmap Index Scan on trading_p2019_02_pkey  (cost=0.00..16.45 rows=6 width=
                    Index Cond: ((trade_ts >= '2019-02-02 00:00:00'::timestamp without time z
Planning Time: 50.683 ms
Execution Time: 0.096 ms
(8 rows)
```

PG 12

```
Aggregate  (cost=25.95..25.96 rows=1 width=8) (actual time=0.022..0.023 rows=1 loops=1)
  ->  Bitmap Heap Scan on trading_p2019_02  (cost=16.45..25.94 rows=6 width=0) (actual
        Recheck Cond: ((trade_ts >= '2019-02-02 00:00:00'::timestamp without time zone)
        ->  Bitmap Index Scan on trading_p2019_02_pkey  (cost=0.00..16.45 rows=6 width=
              Index Cond: ((trade_ts >= '2019-02-02 00:00:00'::timestamp without time z
Planning Time: 0.317 ms
Execution Time: 0.096 ms
(7 rows)
```

Removing [Merge]Append nodes

which contain a single subpath

PG 11

```
Merge Append  (cost=0.43..73.08 rows=1388 width=30) (actual time=0.027..0.419 rows=1440 loops=1)
  Sort Key: trading_p1962.trade_ts
  ->  Index Scan using trading_p1962_trade_ts_idx on trading_p1962  (cost=0.42..59.19 rows=1388 width
        Index Cond: ((trade_ts >= '1962-01-01 00:00:00'::timestamp without time zone) AND (trade_ts <
Planning Time: 35.224 ms
Execution Time: 0.681 ms
```

PG 12

```
Index Scan using trading_p1962_trade_ts_idx on trading_p1962  (cost=0.28..69.43 rows=1440 width=30) (ac
  Index Cond: ((trade_ts >= '1962-01-01 00:00:00'::timestamp without time zone) AND (trade_ts <= '1962-
Planning Time: 0.326 ms
Execution Time: 0.597 ms
```

Concurrent ATTACH PARTITION

- **ATTACH** and ~~**DETACH**~~ is not blocking SELECTSs and Vise Versa
  - No separate syntax for "**CONCURRENTLY**"

CREATION OF NEW PARTITION TO TABLE

```
CREATE TABLE public.trading_p1963_04 PARTITION OF public.trading FOR VALUES FROM ('1963-04-01
00:00:00') TO ('1963-05-01 00:00:00');
```

```
CREATE TABLE public.trading_p1963_04 (like public.trading_p1963_03);
ALTER TABLE trading ATTACH PARTITION trading_p1963_04 for values from ('1963-04-01 00:00:00') to
('1963-05-01 00:00:00');
```

Partition Tree

```
postgres=# select * from pg_partition_tree('ab');
 relid    | parentrelid | isleaf | level
----------+-------------+--------+-------
 ab       |             | f      |     0
 ab_a2    | ab          | f      |     1
 ab_a1    | ab          | f      |     1
 ab_a3    | ab          | f      |     1
 ab_a2_b1 | ab_a2       | t      |     2
 ab_a2_b2 | ab_a2       | t      |     2
 ab_a2_b3 | ab_a2       | t      |     2
 ab_a1_b1 | ab_a1       | t      |     2
 ab_a1_b2 | ab_a1       | t      |     2
 ab_a1_b3 | ab_a1       | t      |     2
 ab_a3_b1 | ab_a3       | t      |     2
 ab_a3_b2 | ab_a3       | t      |     2
 ab_a3_b3 | ab_a3       | t      |     2
```

`pg_partition_root()` - top-most parent
`pg_partition_ancestors()` - ancestor relations

```
select * from pg_partition_root('ab_a3_b1');
select * from pg_partition_ancestors('ab_a3_b1');
```

Bulk Load (COPY) & INSERT

- Bulk load is done using Bulk insert
- Simple pg_dump test shows 31% improvement
- INSERTs takes less locking.

Partitioning - Foreign key references

PG 11

```
postgres=# ALTER TABLE exceptions ADD CONSTRAINT fk_trading_exceptions FOREIGN
KEY (trade_id,trade_ts) REFERENCES trading (trade_id,trade_ts );
ERROR:  cannot reference partitioned table "trading"
```

PG 12

```
postgres=# ALTER TABLE exceptions ADD CONSTRAINT fk_trading_exceptions FOREIGN
KEY (trade_id,trade_ts) REFERENCES trading (trade_id,trade_ts );
ALTER TABLE
```

Other Partition Improvements

- **Partition boundaries can be defined as expression**
  - It is evaluated at the time of creation

General Performance Improvements

real and **double precision** values

- **Output of floating-point numbers uses different algorithm**
  - Database drivers and Applications like pg_dump
- **Speed up**
- **Consistent across platforms.**
- **Caution:**
  1. Output format might change
  2. Default value of **extra_float_digits** is changed and its implications.

Minimal decompression - deTOAST

- **No full decompression of TOAST is required.**
- **Speed up**
  - PostGIS
  - JSON
  - LIKE '...%'

CTE Optimization

- Avoid Materialization of result set
- Better filtering, index usage, number of rows
- old behaviour WITH MATERIALIZED

VACUUM

```
postgres=# VACUUM (INDEX_CLEANUP FALSE,FREEZE TRUE) COMPANY;
VACUUM
```

```
postgres=# ALTER TABLE COMPANY SET ( vacuum_index_cleanup = FALSE);
ALTER TABLE
```

Skipping the index cleanup can speed up the vacuum process. which will be he handy if we do emergency VACUUM FREEZE

```
postgres=# VACUUM (ANALYZE,  SKIP_LOCKED) parted;
```

Skip those partitions of table which is having a non-compatible lock. instead of waiting

New Features

JSON - SQL 2016

- json_path function
- operators
- index support

- jsonb_path_exists(jsonb, jsonpath[, jsonb, bool]),
- jsonb_path_match(jsonb, jsonpath[, jsonb, bool]),
- jsonb_path_query(jsonb, jsonpath[, jsonb, bool]),
- jsonb_path_query_array(jsonb, jsonpath[, jsonb, bool]).
- jsonb_path_query_first(jsonb, jsonpath[, jsonb, bool]).

COPY FROM with WHERE

- **Filtering of records are possible now**

```
Syntax:
COPY table_name [ ( column_name [, ...] ) ]
    FROM { 'filename' | PROGRAM 'command' | STDIN }
    [ [ WITH ] ( option [, ...] ) ]
    [ WHERE condition ]
```

Checksum - pg_checksums

- **PG11:**
  - Dump data, Initialize cluster, reload the data
  - (unofficial repo)
- **PG12**
  - Shutdown, enable checksum, startup

- **Future expectation:** Live changes

Generated Columns

```
column_name data_type GENERATED ALWAYS AS
   ( generation_expr ) STORED
```

- Eliminates unnecessary triggers
- JSON/XML extraction, GIS data, full-text

```
CREATE TABLE candidate (
    candidate_id INT PRIMARY KEY,
    jobs JSONB,
    since text GENERATED ALWAYS AS (
        jsonb_path_query_first(jobs,'$.*.doj')->>0
    ) STORED
  );
```

Collation - deterministic for case- or accent-insensitivity

"The default is true. A deterministic comparison considers strings that are not byte-wise equal to be unequal even if they are considered logically equal by the comparison. PostgreSQL breaks ties using a byte-wise comparison. Comparison that is not deterministic can make the collation be, say, case- or accent-insensitive.

```
CREATE COLLATION case_insensitive (
        provider = icu,
        locale =
'@colStrength=secondary',
        deterministic = false
```

```
CREATE COLLATION ignore_accents (
  provider = icu,
  locale = '@colStrength=primary;colCaseLevel=yes',
  deterministic = false
);
```

```
CREATE TABLE tab1 (name TEXT COLLATE case_insensitive);
INSERT INTO tab1 VALUES ('JOBIN');

postgres=# select * from tab1 where name =
'jobin';
 name
-------
 JOBIN
(1 row)
```

```
CREATE TABLE tab2 (docs TEXT COLLATE ignore_accents);
INSERT INTO tab2 VALUES ('résumé');

postgres=# postgres=# select * from tab2 where
docs = 'resume';
  docs
--------
 résumé
(1 row)
```

Standby database Improvements

Standby Database

- **recovery.conf is dead**

```
postgres=# show recovery_
recovery_end_command       recovery_target           recovery_target_inclusive  recovery_target_name       recovery_target_timeline
recovery_min_apply_delay   recovery_target_action     recovery_target_lsn        recovery_target_time       recovery_target_xid
```

```
postgres=# select name,setting,unit,context from pg_settings where name like 'recovery%';
```

```
     name           | setting | unit |  context
--------------------+---------+------+------------
 primary_conninfo   |         |      | postmaster
 primary_slot_name  |         |      | postmaster
(2 rows)
```

Standby Database - backup tools

pg_basebackup

- creates **standby.signal**
- adds **primary_conninfo** into **postgresql.auto.conf**

Standby Database - Promote though connection

```
postgres=# select pg_is_in_recovery();
 pg_is_in_recovery
-------------------
 t
(1 row)

postgres=# select pg_promote();
 pg_promote
------------
 t
(1 row)

postgres=# select pg_is_in_recovery();
 pg_is_in_recovery
-------------------
 f
(1 row)
```

Standby Database - trigger file

```
postgres=# alter system set promote_trigger_file='trigger.txt';
ALTER SYSTEM
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)

postgres=# show promote_trigger_file;
 promote_trigger_file
----------------------
 trigger.txt
(1 row)
```

LOG: promote trigger file found: trigger.txt
FATAL: terminating walreceiver process due to administrator command
LOG: invalid record length at 1/47A5D198: wanted 24, got 0
LOG: redo done at 1/47A5D160
LOG: last completed transaction was at log
...
LOG: archive recovery complete
LOG: database system is ready to accept connections

- After the promotion standby.signal will be removed by PostgreSQL

Standby Database

- **recovery.signal** and **standby.signal**
  - if **standby.signal** is present, PostgreSQL will start the standby mode
  - else if **recovery.signal** is present, it will start the targeted recovery.
  - recovery proceeds upto **`latest`** **`recovery_target_timeline`** by default
- HA solutions need to be modified
- Backup tools needs modification.
- ~~Primary connection info can be changed on the fly~~
  - ~~ALTER SYSTEM SET../ pg_reload_conf();~~

PSQL Improvements

PSQL improvements

```
postgres=# \pset format
aligned          asciidoc         csv              html             latex            latex-longtable
troff-ms         unaligned        wrapped
```

- **CSV output**
  - \pset format csv
  - --csv option

- **Help with link - \h**
- **Tab completion of CREATE TABLE, CREATE TRIGGER, CREATE EVENT TRIGGER, ANALYZE, EXPLAIN, VACUUM, ALTER TABLE, ALTER INDEX, ALTER DATABASE, and ALTER INDEX ALTER COLUMN**

PSQL improvements

```
postgres=# \dP
                 List of partitioned relations
 Schema |    Name      |  Owner   |       Type         |  Table
--------+--------------+----------+--------------------+--------
-
 public | trading      | postgres | partitioned table  |
 public | trading_pkey | postgres | partitioned index  | trading
```

```
postgres=# \d
 Schema |         Name          |       Type         |  Owner
--------+-----------------------+--------------------+----------
 public | trading               | partitioned table  | postgres
 public | trading_p1963_01      | table              | postgres
 public | trading_p1963_02      | table              | postgres
 public | trading_p1963_03      | table              | postgres
```

Indexing Improvements

Index Improvements

B-Tree

- **Multi column index is more space efficient**
- **Improved performance of Non-unique index (with duplicates)**
  - Vacuum cleans up indexes with duplicate values
- **Index updates are improved for less locking.**

REINDEX CONCURRENTLY

- **Blocking**
- **Workarounds till PG11**
  - pg_idxmaint
- **PG12: REINDEX <index> CONCURRENTLY**
  - REINDEX <TABLE> CONCURRENTLY

```
postgres=# ALTER TABLE orders DROP CONSTRAINT orders_pkey, ADD CONSTRAINT orders_pkey PRIMARY KEY
USING INDEX orders_pkey_new;

ERROR:  cannot drop constraint orders_pkey on table orders because other objects depend on it
DETAIL:  constraint order_dtls_order_id_fkey on table order_dtls depends on index orders_pkey
```

GIST Index

- Just like B-Tree, GIST index support INCLUDE option
- Index only plans

Less WAL generation

“Instead of WAL-logging every modification during the build separately, first build the index **without any WAL-logging**, and make a **separate pass through the index at the end, to write all pages to the WAL**. This significantly reduces the amount of WAL generated, and is usually also faster, despite the extra I/O needed for the extra scan through the index.WAL generated this way is also faster to replay.**”**

- **GiST, GIN and SP-GiST index build and WAL replay improves**

Optimizer Improvements

Better Statistics- Multi-column most-common-value (MCV)

```
CREATE STATISTICS func_deps_stat (dependencies) ON a, b, c FROM functional_dependencies;
```

- New View : pg_stats_ext
- New function : pg_mcv_list_items()

```
CREATE STATISTICS mcv_stat (mcv) ON a, b FROM stats;
```

https://www.postgresql.org/docs/12/functions-statistics.html
https://www.postgresql.org/docs/12/view-pg-stats-ext.html

Prepared statements - Generic vs Custom Plan

- PostgreSQL 12 introduces the capability to control

```
SET plan_cache_mode = 'force_custom_plan';
SET plan_cache_mode = 'force_generic_plan';
SET plan_cache_mode = 'auto';
```

JIT

- JIT is enabled by default.
- It kicks in more often and select clause

```
 Insert on trading  (cost=0.00..2400000.00 rows=30000000 width=38) (actual
time=755568.562..755568.562 rows=0 loops=1)
   ->  Subquery Scan on "*SELECT*"  (cost=0.00..2400000.00 rows=30000000 width=38) (actual
time=4369.507..89923.141 rows=30000000 loops=1)
         ->  Function Scan on generate_series g  (cost=0.00..1650000.00 rows=30000000 width=36)
(actual time=2084.311..23046.387 rows=30000000 loops=1)
 Planning Time: 0.421 ms
 JIT:
   Functions: 5
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 2.388 ms, Inlining 51.787 ms, Optimization 32.930 ms, Emission 24.354 ms,
Total 111.459 ms
 Execution Time: 756540.286 ms
(9 rows)
```

Caution

JIT can be dangerous

```
 JIT:
   Functions: 5480
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 486.285 ms, Inlining 117.954 ms, Optimization 24299.944 ms, Emission
15986.465 ms, Total 40890.648 ms
 Execution Time: 16141.694 ms
(924 rows)

Time: 16176.248 ms (00:16.176)
```

```
postgres=# set jit=off;
```

```
 Planning Time: 21.500 ms
 Execution Time: 2798.233 ms
(920 rows)

Time: 2833.774 ms (00:02.834)
```

Monitoring Improvements

log_transaction_sample_rate:

~~log_statement_sample_rate~~

Set the fraction of transactions whose statements are all logged, in addition to statements logged for other reasons

- **This is the option to capture fast queries**
- **Possible to adjust log_min_duration_statement**
- **Best for query optimization**

Progress Monitoring

PostgreSQL 11:

VACCUM - pg_stat_progress_vacuum

PostgreSQL 12:

CLUSTER, VACUUM FULL - pg_stat_progress_cluster

CREATE INDEX, REINDEX - pg_stat_progress_create_index

Security Improvements

Security - multi-factor authentication

- **clientcert=verify-full**
  - valid certificate
  - cn should match username
- **can be added to existing authentication**

```
hostssl  database     user          IP-CIDR      auth-method  [auth-options]
hostssl   all         testuser      X.X.X.X/XX   password     clientcert=verify-full
```

https://www.postgresql.org/docs/12/ssl-tcp.html#SSL-CLIENT-CERTIFICATES

Security

ssl_min_protocol_version: - Sets the minimum SSL/TLS protocol version to use. Valid values are currently: **TLSv1**, TLSv1.1, TLSv1.2, TLSv1.3

ssl_max_protocol_version:- The default is to allow any version. useful for testing / troubleshooting

- Error will be raised for unsupported version
- Select a latest version of TSL if application / software supports it

Security - GSSAPI Authentication

"GSSAPI is an **industry-standard protocol for secure authentication defined in RFC 2743**. PostgreSQL supports GSSAPI for use as either an **encrypted, authenticated layer, or for authentication only**. GSSAPI provides **automatic authentication (single sign-on)** for systems that support it."

Additional Reference

David Rowley - https://www.2ndquadrant.com/en/blog/postgresql-12-partitioning/

Hans-Jürgen Schönig - https://www.cybertec-postgresql.com/en/tech-preview-improving-copy-and-bulkloading-in-postgresql-12/

Release Notes : https://www.postgresql.org/docs/release/12.0/

Multi-Factor authentication Mail Thread : https://www.postgresql.org/message-id/CABUevEySUALk7Z9wn0baWswS3crfOazH_-nd0O0Pi7nS5SG7TA%40mail.gmail.com

Thank You

Champions of Unbiased
Open Source Database Solutions