POSTGRESQL is an open-source, full-featured relational database. This presentation gives an overview of the Postgres 12 release.

Creative Commons Attribution License http://momjian.us/presentations

Last updated: October, 2019
1. Partitioning improvements
2. Btree improvements
3. Multi-column most-common-value statistics
4. Inline many CTE queries
5. Prepared plan control
6. Just-in-time Compilation
7. Checksum Control
8. REINDEX CONCURRENTLY

Full item list at https://www.postgresql.org/docs/12/index.html
1. Partitioning Improvements

- Thousands of partitions now efficiently processed
- Partitioned tables can now be referenced as foreign keys
- Improve COPY into partitioned tables
- Partition bounds can now be expressions
- New partition introspection SQL functions
2. Btree Improvements

- Reduce multi-column index size by using space more efficiently
- Improve performance of indexes with many duplicates
- Allow `VACUUM` to more efficiently remove tuples from indexes with many duplicates
- Reduce locking requirements during index updates
3. Multi-Column Most-Common-Value Statistics

Allow most-common-value statistics for multiple columns; previously only a single correlation value was recorded for multiple columns.

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

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

<table>
<thead>
<tr>
<th>index</th>
<th>values</th>
<th>nulls</th>
<th>frequency</th>
<th>base_frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{Washington, DC}</td>
<td>{f,f}</td>
<td>0.003467</td>
<td>2.7e-05</td>
</tr>
<tr>
<td>1</td>
<td>{Apo, AE}</td>
<td>{f,f}</td>
<td>0.003067</td>
<td>1.9e-05</td>
</tr>
<tr>
<td>2</td>
<td>{Houston, TX}</td>
<td>{f,f}</td>
<td>0.002167</td>
<td>0.000133</td>
</tr>
<tr>
<td>3</td>
<td>{El Paso, TX}</td>
<td>{f,f}</td>
<td>0.002</td>
<td>0.000113</td>
</tr>
<tr>
<td>4</td>
<td>{New York, NY}</td>
<td>{f,f}</td>
<td>0.001967</td>
<td>0.000114</td>
</tr>
<tr>
<td>5</td>
<td>{Atlanta, GA}</td>
<td>{f,f}</td>
<td>0.001633</td>
<td>3.3e-05</td>
</tr>
<tr>
<td>6</td>
<td>{Sacramento, CA}</td>
<td>{f,f}</td>
<td>0.001433</td>
<td>7.8e-05</td>
</tr>
<tr>
<td>7</td>
<td>{Miami, FL}</td>
<td>{f,f}</td>
<td>0.0014</td>
<td>6e-05</td>
</tr>
<tr>
<td>8</td>
<td>{Dallas, TX}</td>
<td>{f,f}</td>
<td>0.001367</td>
<td>8.8e-05</td>
</tr>
<tr>
<td>9</td>
<td>{Chicago, IL}</td>
<td>{f,f}</td>
<td>0.001333</td>
<td>5.1e-05</td>
</tr>
</tbody>
</table>
Many common table expressions (CTE) can now be inlined:

-- PG 11
EXPLAIN WITH t(x) AS (SELECT 1) SELECT * FROM t;

QUERY PLAN
--------------------------------------------------
CTE Scan on t  (cost=0.01..0.03 rows=1 width=4)
  CTE t
    -> Result  (cost=0.00..0.01 rows=1 width=4)

-- PG 12
EXPLAIN WITH t(x) AS (SELECT 1) SELECT * FROM t;

QUERY PLAN
--------------------------------------------------
Result  (cost=0.00..0.01 rows=1 width=4)
Prepared statements usually use generic/prepared plans after six executions:

```sql
PREPARE p (INTEGER) AS
SELECT relname FROM pg_class WHERE oid = $1;

EXPLAIN EXECUTE p (1);

<table>
<thead>
<tr>
<th>QUERY PLAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Scan using pg_class_oid_index on pg_class...</td>
</tr>
<tr>
<td>Index Cond: (oid = '1'::oid)</td>
</tr>
</tbody>
</table>

EXPLAIN EXECUTE p (1);

<table>
<thead>
<tr>
<th>QUERY PLAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Scan using pg_class_oid_index on pg_class...</td>
</tr>
<tr>
<td>Index Cond: (oid = '1'::oid)</td>
</tr>
</tbody>
</table>
EXPLAIN EXECUTE p (1);

QUERY PLAN

Index Scan using pg_class_oid_index on pg_class ...
  Index Cond: (oid = '1'::oid)

EXPLAIN EXECUTE p (1);

QUERY PLAN

Index Scan using pg_class_oid_index on pg_class ...
  Index Cond: (oid = '1'::oid)
EXPLAIN EXECUTE p (1);               QUERY PLAN

-----------------------------------------------
Index Scan using pg_class_oid_index on pg_class ...
  Index Cond: (oid = '1'::oid)

EXPLAIN EXECUTE p (1);

-----------------------------------------------
Index Scan using pg_class_oid_index on pg_class ...
  Index Cond: (oid = ($1)::oid)
plan_cache_mode allows users to force always-custom or always-generic plans:

```sql
DEALLOCATE p;

PREPARE p (INTEGER) AS SELECT relname FROM pg_class WHERE oid = $1;

SET plan_cache_mode = force_generic_plan;

EXPLAIN EXECUTE p (1);
```

```
QUERY PLAN

Index Scan using pg_class_oid_index on pg_class ...
Index Cond: (oid = ($1):: oid)
```
6. Just-in-Time Compilation

- Enable JIT by default
- Useful for data warehouse queries
7. Checksums Control

- Allow a cluster’s checksum mode to be changed while it is offline
- `pg_checksums --enable --progress`
- Online change control planned
8. **REINDEX CONCURRENTLY**

Like `CREATE INDEX CONCURRENTLY`, this allows `REINDEX` with minimal locking, specifically, just before completion:

```sql
CREATE TABLE test (x INTEGER);

INSERT INTO test SELECT generate_series(1, 1000);

CREATE INDEX i_test ON test (x);

REINDEX INDEX CONCURRENTLY i_test;
```
Conclusion

http://momjian.us/presentations

https://www.flickr.com/photos/thomasletholsen/