Nuevas características de PostgreSQL 11

Álvaro Herrera
alvherre@2ndQuadrant.com

2ndQuadrant Ltd.
https://www.2ndQuadrant.com/

Meetup PostgreSQL Santiago
https://www.postgresql.org/

2ndQuadrant PostgreSQL

Acerca de 2ndQuadrant

- Empresa de servicios y soporte a PostgreSQL
- Auspiciador principal de la comunidad PostgreSQL
- Desarrolladores de características principales de PostgreSQL

2ndQuadrant®
PostgreSQL

Sobre Álvaro Herrera

- Desarrollador de PostgreSQL desde 2002
- Committer de PostgreSQL desde 2005

2ndQuadrant®
PostgreSQL

Introducción a PostgreSQL 11

- Un año de desarrollo desde PostgreSQL 10
- >180 cambios listados en “release notes”
- >2200 commits
- 313 contribuidores
- 5445 archivos, 950.959 SLOC
- 3150 archivos cambiados

2ndQuadrant®
PostgreSQL

La versión 11 de PostgreSQL

- liberada el 8 de noviembre
- “feature freeze” en abril
- Soporte comunitario durante 5 años

2ndQuadrant®
PostgreSQL

Mejoras en el particionamiento

- particionamiento declarativo es nuevo en PostgreSQL 10
- Mejoras en PostgreSQL 11:
  - particionamiento por hash
  - partición por omisión
  - actualización de llave de particionamiento
  - soporte para llaves primarias y foráneas, índices y triggers en tablas particionadas
  - “poda” de particiones en tiempo de ejecución

Particionamiento por hash

```
CREATE TABLE clientes (
   cliente_id bigint NOT NULL,
   nombre_cliente text NOT NULL,
   direccion text,
   pais text
) PARTITION BY HASH (nombre_cliente);
CREATE TABLE clientes_p1 PARTITION OF clientes
   FOR VALUES WITH (MODULUS 4, REMAINDER 0);
CREATE TABLE clientes_p2 PARTITION OF clientes
   FOR VALUES WITH (MODULUS 4, REMAINDER 1);
CREATE TABLE clientes_p3 PARTITION OF clientes
   FOR VALUES WITH (MODULUS 4, REMAINDER 2);
CREATE TABLE clientes_p4 PARTITION OF clientes
   FOR VALUES WITH (MODULUS 4, REMAINDER 3);
```

Particionamiento por hash

```
CREATE TABLE clientes (
   cliente_id bigint NOT NULL,
   nombre_cliente text NOT NULL,
   direccion text,
   pais text
) PARTITION BY HASH (nombre_cliente);
CREATE TABLE clientes_p1 PARTITION OF clientes
   FOR VALUES WITH (MODULUS 4, REMAINDER 0);
CREATE TABLE clientes_p2 PARTITION OF clientes
   FOR VALUES WITH (MODULUS 4, REMAINDER 1);

CREATE TABLE clientes_p3a PARTITION OF clientes
    FOR VALUES WITH (MODULUS 8, REMAINDER 2);
CREATE TABLE clientes_p3b PARTITION OF clientes
    FOR VALUES WITH (MODULUS 8, REMAINDER 6);

CREATE TABLE clientes_p4 PARTITION OF clientes
  FOR VALUES WITH (MODULUS 4, REMAINDER 3);
```

2ndQuadrant PostgreSQL

Partición por omisión

```
CREATE TABLE clientes (
  cliente_id bigint NOT NULL,
  nombre_cliente text NOT NULL,
  direccion text,
  pais text
) PARTITION BY LIST (pais);

CREATE TABLE clientes_us PARTITION OF clientes FOR VALUES IN ('us');
CREATE TABLE clientes_de PARTITION OF clientes FOR VALUES IN ('de');

CREATE TABLE customers_def PARTITION OF customers DEFAULT;
```

Actualización de llave de particionamiento

```
UPDATE clientes
  SET pais = 'fr'
  WHERE cliente_id = 12345;
```

- Convertido internamente a INSERT + DELETE
- (¡cuidado con update/delete concurrente!)

Índices y llaves en tablas particionadas

```
CREATE TABLE clientes (
cliente_id bigint NOT NULL,
nombre_cliente text PRIMARY KEY, -- debe contener llave de particionami
direccion text,
pais text REFERENCES paises
) PARTITION BY HASH (cnombre_cliente);

CREATE INDEX ON clientes (pais);

/* obtiene llave primaria e indices automaticamente: */
CREATE TABLE clientes_p1 PARTITION OF clientes ...
```

Triggers en tablas particionadas

```
CREATE TRIGGER trg AFTER UPDATE OR INSERT ON TABLE clientes
  FOR EACH ROW EXECUTE PROCEDURE verifica_consist_cliente()
```

Poda más rápida

- Exclusión por restricciones es lenta y limitada
- Poda de particiones es tecnología completamente nueva, más avanzada
- Produce un “programa de poda” a partir del WHERE y los límites de cada partición
- Inicialmente, la poda se aplica en tiempo de optimización
  - tal como exclusión por restricciones

Ejemplo de poda

```
EXPLAIN (ANALYZE, COSTS off)
   SELECT * FROM clientes
   WHERE cliente_id = 1234;
                              QUERY PLAN
-----------------------------------------------------------------------
 Append (actual time=0.054..2.787 rows=1 loops=1)
   -> Seq Scan on clientes_2 (actual time=0.052..2.785 rows=1 loops=1)
         Filter: (cliente_id = 1234)
         Rows Removed by Filter: 12570
 Planning Time: 0.292 ms
 Execution Time: 2.822 ms
(6 filas)
```

Ejemplo sin poda

```
SET enable_partition_pruning TO off;
EXPLAIN (ANALYZE, COSTS off)
   SELECT * FROM clientes
   WHERE cliente_id = 1234;
                                QUERY PLAN
-----------------------------------------------------------------------
 Append (actual time=6.658..10.549 rows=1 loops=1)
   -> Seq Scan on clientes_1 (actual time=4.724..4.724 rows=0 loops=1)
         Filter: (cliente_id = 1234)
         Rows Removed by Filter: 24978
   -> Seq Scan on clientes_00 (actual time=1.914..1.914 rows=0 loops=1)
         Filter: (cliente_id = 1234)
         Rows Removed by Filter: 12644
   -> Seq Scan on clientes_2 (actual time=0.017..1.021 rows=1 loops=1)
         Filter: (cliente_id = 1234)
         Rows Removed by Filter: 12570
   -> Seq Scan on clientes_3 (actual time=0.746..0.746 rows=0 loops=1)
         Filter: (cliente_id = 1234)
         Rows Removed by Filter: 12448
   -> Seq Scan on clientes_01 (actual time=0.648..0.648 rows=0 loops=1)
```

Poda en tiempo de ejecución

- La poda de particiones puede aplicarse en tiempo de ejecución
- Muchas consultas se pueden optimizar mejor en este punto
- Dos momentos para aplicar poda:
  - Cuando se entregan los valores de los parámetros (bind)
  - Cuando valores para columnas surgen desde otro nodo

Ejemplo de poda en tiempo de ejecución

```
explain (analyze, costs off, summary off, timing off)
   execute ab_q1 (2, 2, 3);
                      QUERY PLAN
-----------------------------------------------------
 Append (actual rows=0 loops=1)
   Subplans Removed: 6
   -> Seq Scan on ab_a2_b1 (actual rows=0 loops=1)
         Filter: ((a >= $1) AND (a <= $2) AND (b <= $3))
   -> Seq Scan on ab_a2_b2 (actual rows=0 loops=1)
         Filter: ((a >= $1) AND (a <= $2) AND (b <= $3))
   -> Seq Scan on ab_a2_b3 (actual rows=0 loops=1)
         Filter: ((a >= $1) AND (a <= $2) AND (b <= $3))
(8 rows)
```

Otro ejemplo de poda en tiempo de ejecución

```
explain (analyze, costs off, summary off, timing off)
select * from tbl1 join tbl2 on tbl1.col1 < tbl2.col1;
                              QUERY PLAN
-----------------------------------------------------------------------
 Nested Loop (actual rows=1 loops=1)
   -> Seq Scan on tbl1 (actual rows=1 loops=1)
   -> Append (actual rows=1 loops=1)
         -> Index Scan using tbl2_1_idx on tbl2_1 (never executed)
               Index Cond: (tbl1.col1 < col1)
         -> Index Scan using tbl2_2_idx on tbl2_2 (never executed)
               Index Cond: (tbl1.col1 < col1)
         -> Index Scan using tbl2_5_idx on tbl2_5 (never executed)
               Index Cond: (tbl1.col1 < col1)
         -> Index Scan using tbl2_6_idx on tbl2_6 (actual rows=1 loops=1
               Index Cond: (tbl1.col1 < col1)
(15 rows)
```

2ndQuadrant PostgreSQL

Procedimientos almacenados

- Similares a funciones
- no retornan valores
- se invocan con CALL
- todos los lenguajes/PLs están soportados
- pueden iniciar/terminar transacciones
- compatible con DB2, Oracle

Ejemplo de procedimiento almacenado

```
CREATE PROCEDURE nuevo_cliente(nombre text, direccion
text)
LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO clientes VALUES (nombre, direccion);
END
$$;

CALL nuevo_cliente ('su nombre', 'una direccion');
```

Procedimientos con transacciones

```
CREATE PROCEDURE transaction_test1(x int, y text)
LANGUAGE plpgsql
AS $$
BEGIN
    FOR i IN 0..x LOOP
        INSERT INTO test1 (a, b) VALUES (i, y);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END
$$;
```

Ejemplo: UPDATE en lotes

http://www.postgresonline.com/journal/index.php?/archives/390-Using-procedures-for-batch-geocoding-and-other-batch-processing.html

```
CREATE OR REPLACE PROCEDURE batch_geocode()
LANGUAGE plpgsql
AS $$
BEGIN
  WHILE EXISTS (SELECT 1 FROM addr_to_geocode WHERE pt IS NULL) LOOP
    WITH a AS (SELECT addid, address FROM addr_to_geocode WHERE pt IS NU
               ORDER BY addid LIMIT 5 FOR UPDATE SKIP LOCKED)
    UPDATE addr_to_geocode SET pt = ST_SetSRID(g.geomout,4326)::geograph
      FROM (SELECT addid, (gc).rating, (gc).addy, (gc).geomout
              FROM a LEFT JOIN LATERAL geocode(address,1) AS gc ON (true)
            ) AS g
     WHERE g.addid = addr_to_geocode.addid;

     COMMIT;
  END LOOP;
END;
$$;
```

2ndQuadrant PostgreSQL

Índices *covering*

- `CREATE INDEX ON tabla (a, b) INCLUDE (c)`
- Columna extra sirve para recorridos de sólo-índice
- Pueden incluirse columnas sin soporte b-tree (ej. geometrías)
- Recuerde hacer VACUUM

ALTER TABLE .. ADD COLUMN .. DEFAULT

- antiguamente requería reescribir la tabla completa

```
-- PostgreSQL 10
=# alter table t add column c text default 'hola';
ALTER TABLE
Time: 128021,645 ms (02:08,022)
```

- ahora es sólo un cambio en catálogo

```
-- PostgreSQL 11
=# alter table t add column c text default 'hola';
ALTER TABLE
Time: 59,857 ms
```

2ndQuadrant
PostgreSQL

Mejoras al paralelismo

- Consultas en paralelo introducidas en PostgreSQL 9.6
  - recorrido secuencial
  - agregación
- mejoras en PostgreSQL 10
  - recorridos de índice
  - merge join
  - subconsultas
  - mejor configuración
- mejoras en PostgreSQL 11
  - Hash join
  - `CREATE TABLE AS`, `CREATE MATERIALIZED VIEW`
  - `CREATE INDEX` para btree

2ndQuadrant PostgreSQL

Hash join en paralelo

```
EXPLAIN SELECT count(*) FROM r JOIN s USING (id);

                        QUERY PLAN
-----------------------------------------------------------
Finalize Aggregate
  -> Gather
     Workers Planned: 2
     -> Partial Aggregate
        -> Parallel Hash Join
           Hash Cond: (r.id = s.id)
           -> Parallel Seq Scan on r
           -> Parallel Hash
              -> Parallel Seq Scan on s
```

DDL en paralelo

- CREATE TABLE AS
- CREATE MATERIALIZED VIEW
- índices btree en paralelo

- automático
- sujeto a `max_parallel_maintenance_workers`

Compilación JIT

- JIT compila algunas partes de las consultas a código de máquina
- la compilación tiene un costo
  - sólo para consultas grandes (OLAP / BI / DWH)
- requiere LLVM
- desactivado por omisión (`jit=off`)
- `jit_above_cost`

Qué se JIT-compila

- evaluación de expresiones
  - `WHERE a + b > 55`
- deconstrucción de tuplas
  - `clientes.direccion`

Además:

- optimización adicional del compilador (`-O3`)
  - `jit_optimize_above_cost`
- *inline* de funciones
  - `jit_inline_above_cost`

Ejemplo de JIT

```
EXPLAIN ANALYZE SELECT sum(relpages) FROM pg_class;
                                        QUERY PLAN
-----------------------------------------------------------------------------------
Aggregate (cost=16.27..16.29 rows=1 width=8) (actual time=6.049..6.049
  -> Seq Scan on pg_class (cost=0.00..15.42 rows=342 width=4) (actual t
  Planning Time: 0.133 ms
  JIT:
    Functions: 3
    Generation Time: 1.259 ms
    Inlining: false
    Inlining Time: 0.000 ms
    Optimization: false
    Optimization Time: 0.797 ms
    Emission Time: 5.048 ms
  Execution Time: 7.416 ms
```

2ndQuadrant PostgreSQL

Configuración por omisión

```
jit_above_cost = 100000
jit_inline_above_cost = 500000
jit_optimize_above_cost = 500000
```

Mejoras de seguridad

- *Passphrases* en SSL
- se desactivó la compresión en SSL
- enlazado (*binding*) de canal SCRAM
- `ldapsearchfilter`
- `ldaps://`

Más cambios menores

- Soporte para particiones en FDW (insert, update, delete, copy)
- Tamaño de segmento WAL (16 MB) se puede cambiar durante `initdb`
- replicación lógica soporta `TRUNCATE`
- más opciones de *frame* en funciones ventana
- funciones SHA-2
- `psql` reconoce órdenes `exit` y `quit`
- `pg_verify_checksums`

`https://www.postgresql.org/docs/devel/static/release-11.html`

Mirando al futuro

- PostgreSQL 12 ya está en progreso
- rendimiento con tablas particionadas
- mejoras en JIT (`jit=on`)
- motores de almacenamiento
- mejoras de seguridad
- mejoras de HA/replicación
- mucho más

`https://commitfest.postgresql.org/18/` en adelante

2ndQuadrant®
PostgreSQL

Resumen: PostgreSQL 11

- particionamiento
- paralelismo
- procedimientos
- JIT
- y más

2ndQuadrant®
PostgreSQL

¿Preguntas?

- Blog: `https://blog.2ndquadrant.com`
- Sitio web: `https://www.2ndquadrant.com`
- email: `info@2ndquadrant.com`

2ndQuadrant®
PostgreSQL