Partition and Conquer Large Data In PostgreSQL 10

Amit Langote (NTT OSS Center)
Ashutosh Bapat (EnterpriseDB)

PGCon 2017, Ottawa

Who

Amit Langote

- NTT OSS Center
- Co-author of the VACUUM progress reporting feature in 9.6

Ashutosh Bapat

- EnterpriseDB

Outline

- Declarative Partitioning features in PostgreSQL 10
- Partitioning syntax examples and limitations
- Relationship with inheritance
- Why “declarative” partitioning sounds promising
- Partitioning optimizations with declarative partitioning
  - Partition-pruning
  - Partition-wise operations with examples, performance figures

NTT

PostgreSQL 10 Introduces Declarative Partitioning

What does it provide yet

- Native support for range and list partitioning
- Fast tuple routing
- Commands for partition roll-in and roll-out
- Multi-level partitioning
- Creating partitions as foreign tables
- Significantly improved usability

Quick examples


```
CREATE TABLE orders (
    order_id    int,
    order_date  date
) PARTITION BY RANGE (order_date);

CREATE TABLE orders_y17m05
    PARTITION OF orders
    FOR VALUES FROM ('2017-05-01') TO ('2017-06-01');

INSERT INTO orders VALUES (1, '2017-05-01');
INSERT 0 1

SELECT tableoid::regclass AS partition, * FROM orders;
  partition    | order_id | order_date
---------------+----------+------------
orders_y17m05 |     1    | 2017-05-01
(1 row)

INSERT INTO orders VALUES (1, '2017-06-01');
ERROR:  no partition of relation "orders" found for row
```

NTT

Quick examples

```
CREATE TABLE orders_y17m06 (
   LIKE orders
) PARTITION BY RANGE (order_date);

CREATE TABLE orders_y17m06_1
   PARTITION OF orders_y17m06
   FOR VALUES FROM ('2017-06-01') TO ('2017-06-15');

CREATE TABLE orders_y17m06_2
   PARTITION OF orders_y17m06
   FOR VALUES FROM ('2017-06-15') TO ('2017-07-01');

ALTER TABLE orders
   ATTACH PARTITION orders_y17m06
   FOR VALUES FROM ('2017-06-01') TO ('2017-07-01');

INSERT INTO orders VALUES (2, '2017-06-01');
INSERT 0 1
INSERT INTO orders VALUES (3, '2017-06-17');
INSERT 0 1
```

Quick examples

```
SELECT tableoid::regclass AS partition, * FROM orders;
   partition    | order_id | order_date
----------------+----------+------------
orders_y17m05   |        1 | 2017-05-01
orders_y17m06_1 |        2 | 2017-06-01
orders_y17m06_2 |        3 | 2017-06-17
(3 rows)

EXPLAIN (COSTS OFF)
SELECT * FROM orders WHERE order_date < '2017-06-15';
                    QUERY PLAN
-----------------------------------------------
Append
  ->  Seq Scan on orders_y17m05
        Filter: (order_date < '2017-06-15'::date)
  ->  Seq Scan on orders_y17m06_1
        Filter: (order_date < '2017-06-15'::date)
(5 rows)
```

Quick examples

```
\d+ orders_y17m05
      Table "public.orders_y17m05"
  Column    |  Type    | Collation | Nullable
------------+---------+-----------+----------
order_id    | integer |           |
order_date | date     |           | not null
Partition of: orders FOR VALUES FROM ('2017-05-01') TO ('2017-06-01')
Partition constraint: ((order_date >= '2017-05-01'::date) AND
                       (order_date  < '2017-06-01'::date))

ALTER TABLE orders DETACH PARTITION orders_y17m05;

SELECT tableoid::regclass AS partition, * FROM orders;
   partition     | order_id | order_date
-----------------+----------+------------
orders_y17m06_1 |        2 | 2017-06-01
orders_y17m06_2 |        3 | 2017-06-17
(2 rows)
```

What does it NOT provide

- Ability to create indexes (hence constraints like UNIQUE) on partitioned tables
- Automatic creation of partitions for incoming data or even a “default” partition that would capture any data for which no partition is defined
- Moving rows from one partition to another as part of executing an UPDATE statement that modifies the partition key
- Routing tuples to foreign partitions
- Ability to change partitioning of data after-the-fact by “splitting” a partition or by “merging” partitions

Relationship with inheritance

- Partitioning is really a subset of the inheritance model,
  - Although it imposes more constraints on the schema design and provides more information to the system
- Currently uses the same optimizer code as used to perform inheritance planning
  - And hence suffers the same problems as inheritance when using large number of partitions (child tables)
- Partitioning offers information about partitioning in a more suitable format than when using inheritance
  - Makes it possible to implement faster algorithms in the planner for partitioned tables using this information that also scale well
  - Makes it possible to create partition-wise plans

Why is “declarative” partitioning promising

- Because PostgreSQL developers promised so for years? ☺
- More seriously, it establishes a base on which to implement performance and scalability features for storing and accessing large amounts of data using partitioned tables
- “Many” optimizer improvements possible
  - Optimizer will be able to generate plans such that queries over partitioned table(s) accessing large amounts of data can be performed using highly-parallel per-partition units of work and generate such plans much quicker
- Potential for improvements in other areas of the backend code which become bottleneck when using large number of tables (in the form of partitions)
  - For example, it might be possible to implement special scheme of locking for partitions so that the lock manager is not overwhelmed by large number of partitions