Partitioning Improvements in PostgreSQL 11

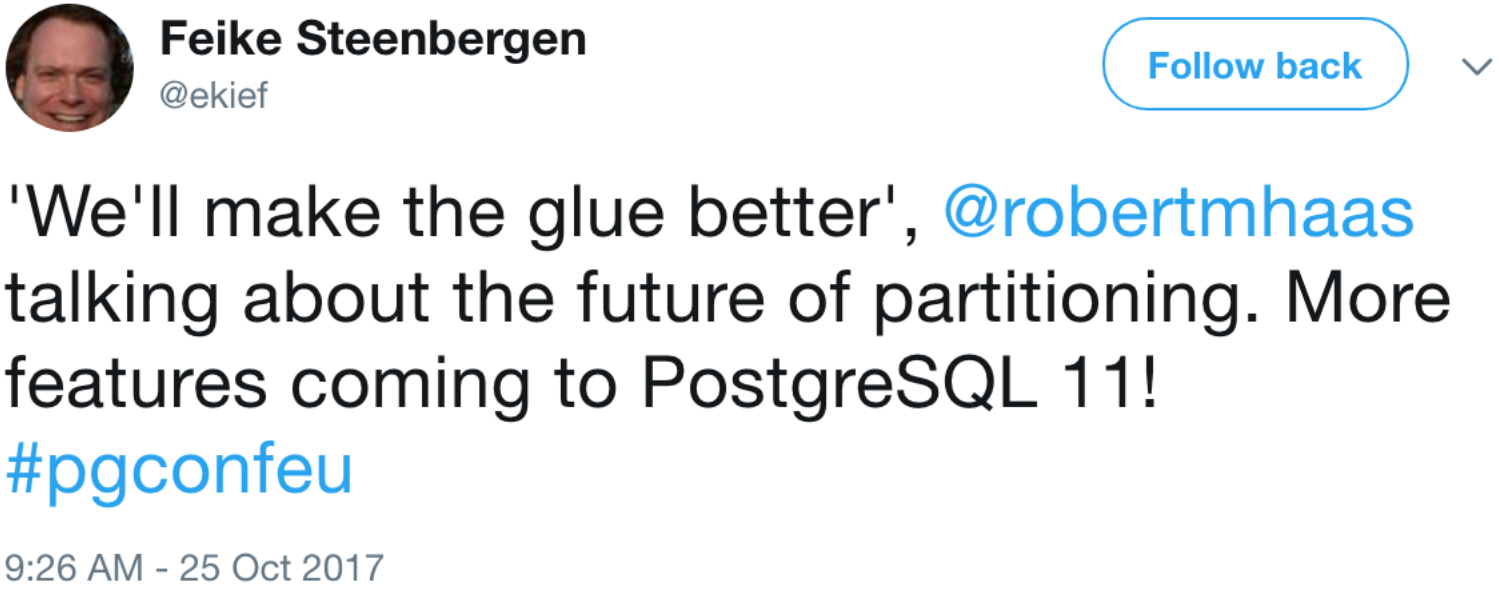
- Robert Haas | 2018-04-18

Overview

- Partitioning as Glue
- Performance Improvements
- New Partitioning Options
- DDL Improvements
- DML Improvements

Partitioning as Glue

- In PostgreSQL, a partitioned table is really just a bunch of individual tables that have been glued together.
- When I gave a talk about partitioning at PGCONF.EU in the fall, I said…



The Glue Is Better

- We've come a long way in six months.
- The glue is definitely still not perfect, but major progress has been made.
- It's still going to be important to keep improving the glue...

Performance Improvements

- Partition pruning is faster.
- Partition pruning can happen at runtime.
- Joins and aggregates can be performed partition-wise.

Partition Pruning Is Faster

PostgreSQL 10 checks each partition to see whether it can be eliminated, but PostgreSQL 11 can directly computed the set of required partitions.

SELECT * FROM foo WHRE a = 314159;

PostgreSQL Version	10 partitions @ 1e6 rows	100 partitions @ 1e6 rows
10.latest	5827 tps	963 tps
master (11devel)	7033 tps (+21%)	1226 tps (+27%)

But It's Not All That Fast

However, it's still vastly slower than just using an unpartitioned table, because pruning happens too late in the process, after too much work has already been done.

master (11devel)	1e7 rows	1e8 rows
unpartitioned	7033 tps	1226 tps
1e6 rows per partition	15221 tps (2.16x)	14883 tps (11.8x)

Runtime Partition Pruning: PREPARE

- v10 and before: If a prepared query gets a custom plan, then partition elimination works, but a generic plan can't eliminate any partitions unless they never need to be scanned for any possible value of the parameter.
- v11: Reconsider the set of partitions that must be scanned at runtime based on the parameter value.
- Example:
  - SELECT * FROM foo WHERE a > $1;

Runtime Partition Pruning: PARAM_EXEC

- SELECT * FROM foo, bar WHERE foo.a = bar.a;
- Both tables have a unique index on column a
- Each table has 10 partitions @ 1e6 rows each
- Forced nested loop with inner index scan
- No parallelism

Runtime Partition Pruning: Plan Shape

```
Aggregate
-> Nested Loop
   -> Append
      -> Seq Scan
      -> Seq Scan
      ...
   -> Append
      -> Index Scan
      -> Index Scan
      ...
```

Runtime Partition Pruning: Results

v10	103 seconds
v11	20 seconds

Partition-wise Join

- If two tables are compatibly partitioned, and being joined on the partitioning columns, we can join each pair of partitions individually.
- In effect, one big join is broken up into a bunch of small joins, improving efficiency.
- Disabled by default out of caution. Set enable_partitionwise_join = true.

Partition-wise Aggregate

- Applies when aggregating a partitioned table, or a partition-wise join of compatibly partitioned tables.
- If the grouping columns include the partitioning columns, can aggregate each partition individually. (Should always win.)
- If not, can perform a PartialAggregate step for each partition and then a FinalizeAggregate step later. (Wins if there are many groups.)
- If the partition is a postgres_fdw foreign table, allows the aggregate to be pushed to the remote side.
- Requires enable_partitionwise_aggregate = true

Partition-wise Join & Aggregate: Before

```
Aggregate
-> Nested Loop
   -> Append
      -> Seq Scan
      -> Seq Scan
      ...
   -> Append
      -> Index Scan
      -> Index Scan
      ...
```

Partition-wise Join & Aggregate: After

```
Finalize Aggregate
-> Append
   -> Partial Aggregate
      -> Nested Loop
         -> Seq Scan
         -> Index Scan
   -> Partial Aggregate
      -> Nested Loop
         -> Seq Scan
         -> Index Scan
   ...
```

Partition-wise Join & Aggregate: Results

v10	103 seconds
v11	20 seconds
v11 + partition-wise	14 seconds

New Partitioning Options

- Hash partitioning is possible.
  - Useful when you want to keep the partition sizes about equal but don't know the distribution of keys in advance.
  - Doesn't provide semantically meaningful partitoins.
- Default partitions are supported for LIST and RANGE partitioning.
  - Keeps inserts from failing; there's always someplace for rows to go.

DDL Improvements

- Partitioned tables can have indexes.
- Partitioned tables can have unique constraints …
  … provided that they include the partition key.
- Partitioned tables can have foreign keys …
  … but they still cannot be the target of a foreign key.
- Partitioned tables can have FOR EACH ROW triggers.

- All of these cases work the same way … a matching object is created for each partition.

DML Improvements

- UPDATE can now move rows between partitions.
- INSERT and COPY can now route rows to foreign partitions.
- INSERT .. ON CONFLICT DO UPDATE/NOTHING.

UPDATE Row Movement: Algorithm

- If the updated tuple still satisfies the partition constraint for the original partition, do the UPDATE normally. Stop here.
- Find the partition that should contain the updated tuple. If there is none, ERROR.
- Insert the updated tuple into that partition.
- Delete the original tuple, marking it as “moved partitions”.
  - Subsequent attempts to update the old version of the tuple will fail with an ERROR instead of being performed on the updated tuple.

Foreign Partitions

- In PostgreSQL 10, foreign tables can be partitions, but you can't route tuples to them.
  - An INSERT or COPY command targeting the partition will work, but if the command targets the partition root, it will fail.
- In PostgreSQL 11, this limitation is removed.
  - Partially works with new UPDATE movement feature: can move tuples to, but not from, a foreign partition.
  - Performance needs work, at least for postgres_fdw.

INSERT .. ON CONFLICT

- DO NOTHING: Just works.
- DO UPDATE: Works provided that there is a matching unique constraint … which can only be true if the unique constraint includes the partitioning columns/expressions.

Thanks

- Any Questions?